A Proofs

This appendix includes all proofs from the paper, providing extra details, for better understanding.

A.1 Complexity

This section of the appendix corresponds to Section 3, where we examine the computational complexity of h^{add}, both in general and specifically for forward generalization. In addition to the detailed proofs, we provide a concrete formalization of the decision problems.

Definition 5. *The decision problem for delete relaxed plan existence on lifted planning tasks is* $\text{LiftedDelRelExist} := \{(\Pi, s) \mid \Pi$ *is a lifted planning task and* s *a state in* Π *for which there exists a delete relaxed plan* $\}$.

Definition 6. *The decision problem for computing* h^{add} *on lifted planning tasks is* $\text{LiftedAdd} := \{(\Pi, s, v) \mid \Pi$ *is a lifted planning task and* s *a state in* Π *and* $h^{add}(s) = v\}$.

Theorem 1. *Computing* h^{add} *on lifted planning tasks is EXPTIME-complete.*

Proof. We start with showing EXPTIME-hard: We reduce from co-LiftedDelRelExist to LiftedAdd. Given an instance (Π, s) return (Π, s, ∞), which is obviously computable in polynomial time. Then Π is not a lifted planning task or s is not a state in Π for which there exists a delete relaxed plan if and only if it holds that Π is not a lifted planning task or s is not a state in Π so that $h^{add}(s) = \infty$, as it is true that for any $s \in S$ there exists no delete relaxed plan in s if and only if $h^{add}(s) = \infty$. Thus LiftedAdd is co-EXPTIME-hard. As LiftedDelRelExist is known to be EXPTIME-hard (Erol, Nau, and Subrahmanian 1995). As EXPTIME = co-EXPTIME, it is also EXPTIME-hard.

Membership in EXPTIME can be checked by the following exponential time algorithm: Ground the task and compute h^{add} on the grounded representation. One can obtain the grounded representation using the naive approach, wherein all variables are replaced by every possible combination of objects in exponential time and so produce an exponentially bounded (w.r.t. to the lifted representation) grounded representation. Further, computing h^{add} runs in polynomial time w.r.t. the grounded representation (Bonet and Geffner 2001), which is exponential w.r.t. the lifted representation. □

Theorem 2. h^{add} *can be computed in polytime on acyclic lifted planning tasks with predicate arity at most* $k \in \mathbb{N}$.

Proof. We consider the task transformatiom (i.e Datalog encoding) from Corrêa et al. (2021) without action predicates. This means that we obtain a delete relaxed planning task (Datalog program) where each action has one add element. Everything expect the actions remains unchanged. The construction for the set of actions goes as follows: Obtain one action (rule) per $(a, p(\vec{x}))$ with $a \in \mathcal{A}, p(\vec{x}) \in add(a)$ where the add list (head) is $\{p(\vec{x})\}$, the precondition (body) is $pre(a)$, the cost is $c(a)$ and $\emptyset$ is the delete list. Note that this encoding is polynomial w.r.t. the lifted planning task representation. This encoding is guaranteed to not change the h^{add} value.

The main observation is that in order to determine which $p(\vec{o}) \in \mathcal{P}^{\mathcal{O}}$ were achieved by a we evaluate the precondition with an outer projection of a finite set of variables. This set corresponds to the variables occurring in each add effect, whose amount is bounded by k. By leveraging acyclicity of the precondition, we can evaluate each result in polynomial time relative to the input, since the output is polynomially bounded by the input, k. (Using Yannakakis algorithm.) Due to the restricted arity of each predicate, there's a limited number of grounded atoms one can assign a cost to. As a result, we establish a uniform cost evaluation to determine newly achievable facts a polynomial amount of times. So given a state $s \in S$ we can use a polynomial time evaluation to determine the cheapest value $h^{add}(s, \{p(\vec{o})\})$ for all $p(\vec{o}) \in \mathcal{P}^{\mathcal{O}}$ and thus determine $h^{add}(s)$.

Note that this proof is very naive, a more realistic realization would be to rewrite the Datalog rules in the lifted forward evaluation according to the join order obtained by the GYO computation. In the resulting program rule has arity at most $2k$ and binary rules. Thus the Datalog evaluation and so h^{add} is performed in polynomial time (Eiter et al. 2007). A similar idea, aligning with this alternative proof, was employed in Corrêa et al. (2023) in the context of grounding a planning task. □

Proposition 3. *For any lifted planning task* Π*, there is an acyclic planning task* Π' *with size at most* $O(|\Pi|)$ *such that any plan in* Π *can be rewritten to a plan in* Π' *and vice versa.*

Proof. We construct Π' from Π. The first step is to make the precondition of each action a with $X(a) = \{x_1, ..., x_n\}$ acyclic by adding an atom $p_a(x_1, ..., x_n)$. This makes the preconditions acyclic as $p_a(x_1, ..., x_n)$ covers all variables. The second steps ensures that all $p_a(x_1, ..., x_n)$, are achievable without explicit enumeration. To do so we add new actions u_{p_a} of cost 0 that make $p_a(x_1, ..., x_n)$ achievable by having one add element $p_a(x_1, ..., x_n)$, but no precondition.

Thus a plan in Π can be converted to a plan in Π' by adding $u_{p_a}(o_1, ..., o_n)$ before each $a(o_1, ..., o_n)$ in the plan. The plan conversion from Π' to Π works by dropping all u_{p_a} actions. □

A.2 Correctness of Formulation and Computational Enhancements

This section of the appendix aligns with Section 5, demonstrating the correctness of the approach, and Section 7, which showcases computational enhancements. We annotate the purpose of intermediate results not found in the paper for better understanding. Please note that the proofs for Section 7 have been move before those for Section 5, as the proofs depend on each other in this order at a technical level.

Theorem 10. *Let* $s \in S$*,* $L \in \mathcal{P}^{X+}$*,* $L_S \subseteq L$ *and* $\text{lext}_S := \{\text{lext}(L, p(\vec{x}), a) \in \text{lext}(L) \mid p(\vec{x}) \in L_S\}$*. If* $s \not\models L_S$*, then:*

$$h^{LRadd}(s, L) = \min_{(a, L') \in \text{lext}_S} c(a) + h^{LRadd}(s, L')$$

Proof. The main idea is to show that some replacement of $p(\vec{x}) \in L_S$ is necessary and the step in which it is performed is irrelevant. Thus it can be enforced immediately. We will

express this formally in the following two conditions that combined imply the correctness of the theorem:

Let $L_0, \ldots, L_n$ so that $L_n = L$ be a lext-path, so that: $L_{i-1} = \text{lext}(L_i, p_i(\vec{x}_i), a_i)$. Condition (1) is completeness: Every lext-path ending in a satisfying set L_0 needs to replace some $p(\vec{x}) \in L_S$. Formally:

$$s \models L_0 \Rightarrow \exists i : 1 \leq i \leq n \wedge p_i(\vec{x}_i) \in L_S$$

and Condition (2) is soundness: If there is a $1 \leq i \leq n$ so that $p_i(\vec{x}_i) \in L_S$ and no $n \geq j > i$ so that $p_i(\vec{x}_i) = p_j(\vec{x}_j)$, then: There exists another lext replacement path $L'_1, \ldots, L'_n$ so that $L'_{i-1} = \text{lext}(L'_i, p'_i(\vec{x}'_i), a'_i)$, $L_n = L'_n$ and the following two conditions hold: (A) $\sum_i c(a_i) = \sum_i c(a'_i)$ and (B) there is a variable remapping $\alpha : X \to X$: so that $L_0 = \alpha(L'_0)$.

We start with proving condition (1): Proof by contradiction. Assume there is no $1 \leq i \leq n$ so that $p_i(\vec{x}_i) \in L_S$. Then all elements $L_S \subseteq L = L_n$ are still in L_0. I.e. $L_S \subseteq L_0$, which implies that $s \not\models L_0$. A contradiction.

Now we continue with proving condition (2): We will prove that we can construct L'_n by simply forcing the replacement $(p_i(\vec{x}_i), a_i)$ to be the first performed replacement. This is we define a bijection $\delta : \{1, \ldots, n\} \to \{1, \ldots, n\}$ to formalize this change in replacement order. We set $\delta(n) = i$, $\delta(j) = j - 1$ for $n > j > i$ and $\delta(j) = j$ otherwise. Further, we set $a_j = a'_{\delta(j)}$ for all $0 \leq j \leq n$, which already guarantees that the actions used for extraction and associated cost are the same (Condition A). We want to perform a similar assignment for the replaced atoms. I.e. that $p'_j(\vec{x}'_j) = p_{\delta(j)}(\vec{x}_{\delta(j)})$ for $0 \leq j \leq n$. Though this is not generally possible, as the variable remapping may change depend on the replacement and so it may occur that $p_{\delta(j)}(\vec{x}_{\delta(j)}) \notin L'_{\delta(j)}$.

In the following we change the selection of the variable remapping function selected from VarRemap in each lext step to allow this assignment. We will show at the end of this proof that all proven properties under this construction generalizes to any choice for VarRemap. Let $1 \leq j \leq n$. Define the mapping $\sigma_j \in \text{VarRemap}(L_j, p_j(\vec{x}_j), a_j)$ used in $\text{lext}(L_i, p_j(\vec{x}_j), a_j)$ as $\sigma_j(x) = x$ for $x \in X(p_j(\vec{x}_j))$ and $\sigma_j(x) = \langle x, p_j(\vec{x}_j), a_j, |\{L_k \mid j < k \leq n, p_j(\vec{x}_j) = p_k(\vec{x}_k), a_j = a_k\}|\rangle$ for $x \notin X(p_j(\vec{x}_j))$. And analogously the mapping $\sigma'_j \in \text{VarRemap}(L'_j, p'_j(\vec{x}'_j), a'_j)$ used in $\text{lext}(L'_j, p'_j(\vec{x}'_j), a'_j)$ as $\sigma'_j(x) = x$ for $x \in X(p'_j(\vec{x}'_j))$ and $\sigma'_j(x) = \langle x, p'_j(\vec{x}'_j), a'_j, |\{L'_k \mid j < k \leq n, p'_j(\vec{x}'_j) = p'_k(\vec{x}'_k), a'_j = a'_k\}|\rangle$ for $x \notin X(p'_j(\vec{x}'_j))$.

The main idea is that we encode the replacement path that derives an atom into the atom itself. This is ensured by: $\langle x, p_j(\vec{x}_j), a_j\rangle$. This part solely depends on the replacements $(p_j(\vec{x}_j), a_j)$, and nothing else. Making the variables name independent of the sets L_j in which the replaced atom occurred. Thus this part will not change variable names when reordering by δ. Enabling the assignment $p'_j(\vec{x}'_j) = p_{\delta(j)}(\vec{x}_{\delta(j)})$ for $0 \leq j \leq n$.

To ensure that $\sigma_j \in \text{VarRemap}(L_j, p_j(\vec{x}_j), a_j)$ is well-defined, we append $|\{L_k \mid j < k \leq n, p_j(\vec{x}_j) = p_k(\vec{x}_k), a_j = a_k\}|$. This guarantees that the variable did not appear in any preceding set. While this adds some dependency on the sets $p(\vec{x})_i$, this values are independent of our reordering. This satisfies the second criterion (2) of VarRemap. Additionally, the condition $\sigma_j(x) = x$ for $x \in X(p_j(\vec{x}_j))$ ensures the first criterion (1) of VarRemap, establishing that the function is well-defined.

$$
\begin{aligned}
& L_0 && \text{[Recursive application of def. lext and } L_i \text{]} \\
&= L_n \setminus p_n(\vec{x}_n) \cup \sigma_n(pre(a_n)) \ldots \setminus p_1(\vec{x}_1) \cup \sigma_1(pre(a_1)) \\
&\text{[count} > 0 \text{ on removal, valid reorder under multiset semantics]} \\
&= L_n \cup \sigma_n(pre(a_n)) \ldots \cup \sigma_1(pre(a_1)) \setminus p_n(\vec{x}_n) \ldots \setminus p_1(\vec{x}_1) \\
&\text{[count} > 0 \text{ on removal, valid reorder under multiset semantics]} \\
&= L_n \setminus p_i(\vec{x}_i) \cup \sigma_i(pre(a_i)) \\
&\qquad \setminus p_n(\vec{x}_n) \cup \sigma_n(pre(a_n)) \\
&\qquad \ldots \\
&\qquad \setminus p_{i+1}(\vec{x}_{i+1}) \cup \sigma_{i+1}(pre(a_{i+1})) \\
&\qquad \setminus p_{i-1}(\vec{x}_{i-1}) \cup \sigma_{i-1}(pre(a_{i-1})) \\
&\qquad \ldots \\
&\qquad \setminus p_1(\vec{x}_1) \cup \sigma_n(pre(a_1)) && \text{[Def. } \delta \text{]} \\
&= L_n \setminus p_{\delta(n)}(\vec{x}_{\delta(n)}) \cup \sigma_{\delta(n)}(pre(a_{\delta(n)})) \\
&\qquad \ldots \\
&\qquad \setminus p_{\delta(1)}(\vec{x}_{\delta(1)}) \cup \sigma_{\delta(1)}(pre(a_{\delta(1)})) && \text{[Def. } a'_i, p'_i, \sigma'_i, \vec{x}'_i \text{]} \\
&= L_n \setminus p'_n(\vec{x}'_n) \cup \sigma'_n(pre(a'_n)) \ldots \setminus p'_1(\vec{x}'_1) \cup \sigma'_1(pre(a'_1)) \\
&= L'_0 && \text{[Recursive application of def. lext and } L'_i \text{]}
\end{aligned}
$$

Figure 4: Rewriting of L_0 into L'_0 in Theorem 10.

We conclude from the rewriting in Fig. 4 that under definition of σ_j, σ'_j it holds that $L_0 = L'_0$, i.e. condition (B) is fulfilled with $\alpha = id$. It is an easy observation that this implicitly allows to build α under an arbitrary selection for σ_j, σ'_j. The main idea is that the restriction of these functions over variables are bijections and step-wise compose α. □

Theorem 11. *Let $L_1, L_2 \in \mathcal{P}^{X+}$, $L = L_1 \cup L_2$ and $X(L_1) \cap X(L_2) = \emptyset$, then $h^{LRadd}(s, L_1) + h^{LRadd}(s, L_2) = h^{LRadd}(s, L)$.*

Proof. For $h^{LRadd}(s, L) = \infty$, the statement holds trivially. Since $X(L_1) \cap X(L_2) = \emptyset$, any replacement of L will also preserve the disjoint variables for replacements arising from . L_1 and L_2 respectively. Hence, if no replacement of L is satisfiable, then at least one of L_1 or L_2 (and their respective replacements) must also be unsatisfiable.

Thus assume $h^{LRadd}(s, L) \neq \infty$. We prove the statement via induction over the recursion depth of h^{LRadd}. For the base case (depth 0), L is satisfied, thus: $h^{LRadd}(s, L) = 0 = h^{LRadd}(s, L_1) + h^{LRadd}(s, L_2)$. Continue with induction step: W.l.o.g. assume $s \not\models L_1$.

$$
\begin{aligned}
&\text{[Definition } h^{LRadd} \text{]} && h^{LRadd}(s, L_1) + h^{LRadd}(s, L_2) \\
&&= \min_{(a, L'_1) \in \text{lext}(L_1)} c(a) + h^{LRadd}(s, L'_1) + h^{LRadd}(s, L_2) \\
&\text{[I.H.]} && = \min_{(a, L'_1) \in \text{lext}(L_1)} c(a) + h^{LRadd}(s, L'_1 \cup L_2) \\
&\text{[Theorem 10]} && = h^{LRadd}(s, L_1 \cup L_2) \\
&\text{[} L_1 \cup L_2 = L \text{]} && = h^{LRadd}(s, L)
\end{aligned}
$$

□

Lemma 6. $\forall s \in S, G \in \mathcal{P}^{\mathcal{O}+}$:
$$h^{LRadd}(s, G) = \sum_{p(\vec{o}) \in G} h^{LRadd}(s, \{p(\vec{o})\})$$

Proof. It holds that $X(G) = \emptyset$ and thus for any $p_1(\vec{o}_1), p_2(\vec{o}_2) \in G$ it holds that $X(p_1(\vec{o}_1)) \cap X(p_2(\vec{o}_2)) = \emptyset$. Thus the claim follows from Theorem 11. □

Proposition 4. $\forall s \in S : h^{add}(s) = \infty \Rightarrow h^{LRadd}(s) = \infty$

Proof. We prove the equivalent statement $h^{LRadd}(s) \neq \infty \Rightarrow h^{add}(s) \neq \infty$ for $s \in S$. In the following we prove that if $h^{LRadd}(s) \neq \infty$ implies the existence of a delete relaxed plan. The existence of a delete relaxed plan is know to be equivalent to $h^{add}(s) \neq \infty$, which concludes the proof. If $h^{LRadd}(s) \neq \infty$, then there is a sequence of multisets in $\mathcal{P}^{X+}$, $L_0, \ldots, L_n$ such that $L_n = \Gamma$, $L_{i-1} = \text{lext}(L_i, p(\vec{x})_i, a_i)$ (using σ_i), and $s \models L_0$ (with $\theta_0 \in \text{match}(s, L_0)$). In the following we show that the sequence $\theta_0(\sigma_1(a_1)), \ldots, \theta_{n-1}(\sigma_{n-1}(a_n))$ with $\theta_i \supseteq \theta_{i-1|X(\theta_i)}$ for $i \in \mathbb{N}^+$ is a delete-relaxed plan for s. (Here $\theta_i \supseteq \theta_{i-1|X(\theta_i)}$ denotes the domain restricted version of θ_{i-1} over the variables $X(\theta_i)$.)

Let $s_0, ..., s_n$ be a sequence of states so that $s_0 = s$ and $s_i = \text{progr}(s_{i-1}, \theta_{i-1}(\sigma_i(a_i^+)))$, i.e. the delete relaxed progression of $\theta_{i-1}(\sigma_i(a_i^+))$ in s_{i-1}. In the following we show for $1 \leq i \leq n$ that s_i is well defined in the sense that actions $\theta_{i-1}(\sigma_i(a_i^+))$ are applicable in s_{i-1}. To do so we additionally need to observe that $\theta_i \in \text{match}(s_i, L_i)$ for $1 \leq i \leq n$. For $i = 1$ we observe that by our construction it holds that $pre(\sigma_1(a_1^+)) \subseteq L_0$. And by definition it holds that $\theta_0 \in \text{match}(s_0, L_0)$. This implies that $pre(\theta_0(\sigma_1(a_1^+))) \subseteq s_0$. For $1 < i \leq n$: Using an inductive argument we can observe that $\theta_{i-1}(\sigma_i(a_i^+))$ is applicable in s_{i-1}. We further observe:

$$\begin{aligned}
& s_i && \text{[Def. } s_i \text{]} \\
&= \text{progr}(s_{i-1}, \theta_{i-1}(\sigma_i(a_i^+))) && \text{[Def. progr, } a^+ \text{]} \\
&= s_{i-1} \cup add(\theta_{i-1}(\sigma_i(a_i))) && \text{[} s_{i-1} \supseteq \theta_{i-1}(L_{i-1}) \text{]} \\
&\supseteq \theta_{i-1}(L_{i-1}) \cup add(\theta_{i-1}(\sigma_i(a_i))) && \text{[Def. } L_{i-1} \text{]} \\
&= \theta_{i-1}(L_i \setminus \{p_i(\vec{x}_i)\} \cup \sigma_i(pre(a_i))) \cup add(\theta_{i-1}(\sigma_i(a_i))) && \text{[Def. } p_i(\vec{x}_i) \text{]} \\
&\supseteq \theta_{i-1}(L_i \setminus add(\sigma_i(a_i)) \cup \sigma_i(pre(a_i))) \cup\ add(\theta_{i-1}(\sigma_i(a_i))) && \text{[} \setminus add(\sigma_i(a_i)) \text{ is obsolete by } \cup add(\sigma_i(a_i)) \text{]} \\
&= \theta_{i-1}(L_i \cup \sigma_i(pre(a_i))) \cup\ add(\theta_{i-1}(\sigma_i(a_i))) && \text{[} A \cup B \supseteq A \text{]} \\
&\supseteq \theta_{i-1}(L_i)
\end{aligned}$$

Thus we observe that there is a $\theta_i \in \text{match}(\theta_i, s_i)$. Note that we need the construction of $\theta_i \supseteq \theta_{i-1}$ as potentially $X(\theta_{i-1}) \not\supseteq X(p_i(\vec{x}_i))$. Further, $pre(\theta_i(\sigma_{i+1}(a_{i+1}^+)))$, thus $\theta_i(\sigma_{i+1}(a_{i+1}^+))$ is applicable in s_i.

Recall that $\theta_n \in \text{match}(s_n, L_n)$ is defined as $s_n \supseteq \theta_n(L_n) = \theta_n(\Gamma) = \Gamma$. This concludes that our construction is a delete relaxed plan. □

The following Lemma describes that for a given lifted atom it is the same to ground the atom and then determine the achievers as to apply and the lifted regression and then ground.

$$\begin{aligned}
& \{\theta(L') \mid (a, L') \in \text{lext}(L)\} && \text{[Def. lext]} \\
&= \{\theta(L') \mid \bot \neq \text{lext}(L, p(\vec{x}), a) = L', a \in \mathcal{A}, p(\vec{x}) \in L\} && \text{[Def. lext]} \\
&= \{\theta(L') \mid L' = L \setminus \{p(\vec{x})\} \cup \sigma(pre(a)), \\
&\quad \sigma \in \text{VarRemap}(L, p(\vec{x}), a), a \in \mathcal{A}, p(\vec{x}) \in L\} && \text{[Redefine } L' \text{ equivalenetly by moving application of } \theta \text{]} \\
&= \{L' \mid L' = \theta(L \setminus \{p(\vec{x})\} \cup \sigma(pre(a))), \\
&\quad \sigma \in \text{VarRemap}(L, p(\vec{x}), a), a \in \mathcal{A}, p(\vec{x}) \in L\} && \text{[Applying } \theta \text{ to operands doesn't change multiset result]} \\
&= \{L' \mid L' = \theta(L) \setminus \{\theta(p(\vec{x}))\} \cup \theta(\sigma(pre(a))), \\
&\quad \sigma \in \text{VarRemap}(L, p(\vec{x}), a), a \in \mathcal{A}, p(\vec{x}) \in L\} && \text{[Def. VarRemap, selection omitted]} \\
&= \{L' \mid L' = \theta(L) \setminus \{\theta(p(\vec{x}))\} \cup \theta(\sigma(pre(a))), \\
&\quad \sigma : X(a) \to X \cup \mathcal{O}, p(\vec{x}) \in add(\sigma(a)), \\
&\quad X(\sigma(a)) \cap X(L) = X(p(\vec{x})), a \in \mathcal{A}, p(\vec{x}) \in L\} && \text{[Observation 1]} \\
&= \{L' \mid L' = \theta(L) \setminus \{\theta(p(\vec{x}))\} \cup \sigma(pre(a)), \\
&\quad \sigma : X(a) \to X \cup \mathcal{O}, \theta(p(\vec{x})) \in add(\sigma(a)), \\
&\quad X(\sigma(a)) \cap X(\theta(L)) = X(p(\vec{x})), a \in \mathcal{A}, p(\vec{x}) \in L\} && \text{[Redfine } p(\vec{x}) \text{ equivalently by applying } \theta \text{ over } L \text{ directly]} \\
&= \{L' \mid L' = \theta(L) \setminus \{p(\vec{x})\} \cup \sigma(pre(a)), \\
&\quad \sigma : X(a) \to X \cup \mathcal{O}, p(\vec{x}) \in add(\sigma(a)), \\
&\quad X(\sigma(a)) \cap X(\theta(L)) = X(p(\vec{x})), a \in \mathcal{A}, p(\vec{x}) \in \theta(L)\} && \text{[Def. VarRemap, selection omitted]} \\
&= \{L' \mid L' = \theta(L) \setminus \{p(\vec{x})\} \cup \sigma(pre(a)), \\
&\quad \sigma \in \text{VarRemap}(\theta(L), p(\vec{x}), a), a \in \mathcal{A}, p(\vec{x}) \in \theta(L)\} && \text{[Def. lext]} \\
&= \{L' \mid \bot \neq \text{lext}(\theta(L), p(\vec{x}), a) = L', a \in \mathcal{A}, p(\vec{x}) \in \theta(L)\} && \text{[Def. lext]} \\
&= \{L' \mid (a, L') \in \text{lext}(\theta(L))\}
\end{aligned}$$

Figure 5: Rewriting of lext application with θ in Lemma 13.

Lemma 12. *Let $p(\vec{x}) \in \mathcal{P}^X$ be a lifted atom, then:*

$$\{\theta'(a) \mid \theta'(a) \in \text{ach}(\theta(\{p(\vec{x})\})), \theta \in \text{match}(\{p(\vec{x})\})\}$$
$$= \{\theta'(a) \mid (a, pre(a)) \in \text{lext}(\{p(\vec{x})\}), \theta' \in \text{match}(a)\}$$

Proof. The first step is to observe that

$$\theta(p(\vec{x})) \in add(\theta'(a)), \theta \in \text{match}(\{p(\vec{x})\})$$
$$\Leftrightarrow \exists \sigma \in \text{VarRemap}(\{p(\vec{x})\}, p(\vec{x}), a)$$

Which can be observed to be true by setting $\theta' = \theta\sigma$. Then conclude:

$$\begin{aligned}
& \theta'(a) \in \text{ach}(\theta(\{p(\vec{x})\})), \theta \in \text{match}(\{p(\vec{x})\}) \\
\Leftrightarrow\ & \theta(p(\vec{x})) \in add(\theta'(a)), \theta \in \text{match}(\{p(\vec{x})\}), \\
& a \in \mathcal{A}, \theta' \in \text{match}(a) && \text{[Definition match]} \\
\Leftrightarrow\ & \exists \sigma \in \text{VarRemap}(\{p(\vec{x})\}, p(\vec{x}), a), \\
& a \in \mathcal{A}, \theta' \in \text{match}(a) && \text{[Observation]} \\
\Leftrightarrow\ & (a, pre(a)) \in \text{lext}(\{p(\vec{x})\}), \theta' \in \text{match}(a) && \text{[Def. lext]}
\end{aligned}$$

□

This following lemma states that if we impose extra restrictions on variables by replacing them, the h^{LRadd} value can only increase.

Lemma 13. *Let $s \in S$, $L \in \mathcal{P}^{X+}$, $\theta \subseteq \theta' \in \text{match}(L)$, and $h^{LRadd}(s, L) \neq \infty$ then:*

$$h^{LRadd}(s, L) \leq h^{LRadd}(s, \theta(L))$$

Proof. All functions $\sigma : X(a) \to X \cup \mathcal{O}$ in the following are derived from the definition of VarRemap. Strictly speaking, we require additional constraints on σ to satisfy a fixed selection criteria. However, for brevity and clarity, omit this in our proofs. It is implied that this condition could be included, and the proofs would remain valid. We provide annotations in the according rewriting steps.

For the actual proof, let us start with two immediate observations. First observe (Observation 1) that for $a \in \mathcal{A}$:

$$\begin{aligned}
& \{\theta(\sigma(pre(a))) \mid \sigma : X(a) \to X \cup \mathcal{O}, p(\vec{x}) \in add(\sigma(a)), \\
& \qquad X(\sigma(a)) \cap X(L) = X(p(\vec{x}))\} \\
=\ & \{\sigma(pre(a)) \mid \sigma : X(a) \to X \cup \mathcal{O}, \theta(p(\vec{x})) \in add(\sigma(a)) \\
& \qquad X(\sigma(a)) \cap X(\theta(L)) = X(p(\vec{x}))\}
\end{aligned}$$

as it holds for both cases that (A) all $x \in X(pre(a)) \cap X(add(a))$ are remapped according to θ and the variables remaining $p(\vec{x})$. And (B) the other variables are remapped according to an arbitrary $\sigma : X(a) \to X \cup \mathcal{O}$.

This allows the second observation (Observation 2), that:

$$\min_{(a,L') \in \text{lext}(L)} c(a) + h^{LRadd}(s, \theta(L'))$$
$$= \min_{(a,L') \in \text{lext}(\theta(L))} c(a) + h^{LRadd}(s, L')$$

which can be observed by rewriting as in Figure 5.

If the recursion depth is 0, then $s \models \theta(L)$ and thus $s \models L$ which is witnessed by a (potential) extension θ. Thus $h^{LRadd}(s, \theta(L)) = 0 = h^{LRadd}(s, L)$. Induction step:

$$\begin{aligned}
& = h^{LRadd}(s, L) \\
& = \min_{(a,L') \in \text{lext}(L)} c(a) + h^{LRadd}(s, L') && \text{[Def } h^{LRadd} \text{]} \\
& \leq \min_{(a,L') \in \text{lext}(L)} c(a) + h^{LRadd}(s, \theta(L')) && \text{[I.H.]} \\
& = \min_{(a,L') \in \text{lext}(\theta(L))} c(a) + h^{LRadd}(s, L') && \text{[Observation 2]} \\
& = h^{LRadd}(s, \theta(L)) && \text{[Def. } h^{LRadd} \text{]}
\end{aligned}$$

□

Lemma 5. $\forall s \in S$, $L \in \mathcal{P}^{X+}$:
$h^{LRadd}(s, L) = \min_{\theta \in \text{match}(L)} h^{LRadd}(s, \theta(L))$

Proof. $h^{LRadd}(s, L) = \infty$ follows directly from Lemma 13. Thus assume $h^{LRadd}(s, L) \neq \infty$. We distinguish the two cases:

(1) $h^{LRadd}(s, L) \leq \min_{\theta \in \text{match}(L)} h^{LRadd}(s, \theta(L))$

and

(2) $h^{LRadd}(s, L) \geq \min_{\theta \in \text{match}(L)} h^{LRadd}(s, \theta(L))$.

Case (1) holds if and only if for all $\theta \in \text{match}(L)$ it holds that $h^{LRadd}(s, L) \leq h^{LRadd}(s, \theta(L))$, which was proven in Lemma 13.

For case (2), as $h^{LRadd}(s, L) \neq \infty$, there is a lext-path $L_0, \ldots, L_n$ so that $L_n = \Gamma$, $L_{i-1} = \text{lext}(L_i, p(\vec{x})_i, a_i)$, $\sum_i c(a_i) = h^{LRadd}(s, L)$ and $s \models L_0$. Then there exists a θ^* under which $s \models L_0$.

The main proof idea is to use that $h^{LRadd}(s, \theta^*(L)) \geq \min_{\theta \in \text{match}(L)} h^{LRadd}(s, \theta(L))$. To formally show this we construct two other variable replacement function depending on θ^*. We set $\theta^- := \pi_{X(\theta^*) \cap X(L)}(\theta^*)$ i.e. this is the restriction of θ^* over variables occurring in L and θ^*. And let $\theta \in \text{match}(L)$ with $\pi_{X(\theta^-)}(\theta) = \theta^-$ i.e. the extension of θ^- to all variables occurring in L. Then we conclude:

$$\begin{aligned}
& h^{LRadd}(s, L) \\
& = h^{LRadd}(s, \theta^*(L)) && \text{[Lemma 13]} \\
& = h^{LRadd}(s, \theta^-(L)) && \text{[} X(\theta^*) \setminus X(L) \text{ irrelevant]} \\
& \geq h^{LRadd}(s, \theta(L)) && \text{[Lemma 13]} \\
& \geq \min_{\theta \in \text{match}(L)} h^{LRadd}(s, \theta(L)) && \text{[} \theta \in \text{match}(L) \text{]}
\end{aligned}$$

□

The following lemma is a more general statement that implies $\forall s \in S : h^{add}(s) = h^{LRadd}(s)$ and is simpler to prove.

Lemma 14. *Let $s \in S$, $L \in \mathcal{P}^{X+}$, then It holds that:*

$$h^{LRadd}(s, L) = \min_{\theta \in \text{match}(L)} h^{add}(s, \theta(L))$$

Under $\min_{\theta \in \text{match}(L)} h^{add}(s, \theta(L)) \neq \infty$.

Proof. Note that relating h^{LRadd} and h^{add} in this way is a bit difficult, as h^{add} is defined over sets and L is a multiset. Though, we can overlook this by our assumption of distinct precondition groundings which ensures that the groundings of a multiset with unique elements and a set, with the same elements, are the same.

Proof via induction over the minimal recursion depth of any $h^{add}(s, \theta(L))$ computation.

Base case: With recursion depth 0 we know there is a $\theta \in \text{match}(L)$ so that $\theta(L) \subseteq s$ and thus $s \vDash \theta(L)$. This implies $s \vDash L$ and $h^{LRadd}(s, L) = 0$.

Induction step: We will distinguish between $|L| = 1$ and $|L| > 1$. $|L| = 0$ is covered by the base case. We will start with showing $|L| = 1$ which basically corresponds to the result from Lemma 12.

$$
\begin{aligned}
&\min_{\theta \in \text{match}(L)} h^{add}(s, \theta(L)) && [\text{ Def. } h^{add} \text{ }] \\
&= \min_{\theta \in \text{match}(L)} \min_{\theta'(a) \in \text{ach}(\theta(L))} c(a) + h^{add}(s, pre(\theta'(a))) \\
& && [\text{ Lemma 12 }] \\
&= \min_{(a, pre(a)) \in \text{lext}(L), \theta' \in \text{match}(a)} c(a) + h^{add}(s, pre(\theta'(a))) \\
& && [\text{ split min, apply } \theta' \text{ over set }] \\
&= \min_{(a, pre(a)) \in \text{lext}(L)} c(a) + \min_{\theta' \in \text{match}(a)} h^{add}(s, \theta'(pre(a))) \\
&= \min_{(a, pre(a)) \in \text{lext}(L)} c(a) + h^{LRadd}(s, pre(a)) && [\text{ I.H. }] \\
&= h^{LRadd}(s, L) && [\text{ Def. } h^{LRadd} \text{ }]
\end{aligned}
$$

Now we will show the case $|L| > 1$ which corresponds to the result from Corollary 6.

$$
\begin{aligned}
&\min_{\theta \in \text{match}(L)} h^{add}(s, \theta(L)) \\
&= \min_{\theta \in \text{match}(L)} \sum_{p(\vec{x}) \in L} h^{add}(s, \theta(\{p(\vec{x})\})) && [\text{ Def. } h^{add} \text{ }] \\
&= \min_{\theta \in \text{match}(L)} \sum_{p(\vec{x}) \in L} \min_{\theta' \in \text{match}(\theta(\{p(\vec{x})\}))} h^{add}(s, \theta'(\theta(\{p(\vec{x})\}))) \\
& && [\text{ match}(\theta(\{p(\vec{x})\})) \text{ irrelevant }] \\
&= \min_{\theta \in \text{match}(L)} \sum_{p(\vec{x}) \in L} h^{LRadd}(s, \theta(\{p(\vec{x})\})) && [\text{ I.H. }] \\
&= \min_{\theta \in \text{match}(L)} h^{LRadd}(s, \theta(L)) && [\text{ Cor. 6 }] \\
&= h^{LRadd}(s, L) && [\text{ Lemma 5 }]
\end{aligned}
$$

□

Theorem 7. $\forall s \in S : h^{add}(s) = h^{LRadd}(s)$

Proof. The case $h^{add}(s) = \infty$ is proven in Proposition 4. For $h^{add}(s) \neq \infty$: The rewriting argument, as presented in the main paper, is proven within Lemma 14. Thus we conclude:

$$
\begin{aligned}
&h^{add}(s) \\
&= h^{add}(s, \Gamma) && [\text{ Def. } h^{add} \text{ }] \\
&= \min_{\theta \in \text{match}(\Gamma)} h^{add}(s, \theta(\Gamma)) && [\text{ } \Gamma \text{ grounded }] \\
&= h^{LRadd}(s, \Gamma) && [\text{ Lemma 14 }] \\
&= h^{LRadd}(s) && [\text{ Def. } h^{LRadd} \text{ }]
\end{aligned}
$$

□

A.3 Bounding h^{add}

For completeness here we provide a bound on h^{add}. The bound guarantees theoretical soundness but does not help in practice. We're willing to accept this for now and consider it an exciting avenue for future work.

Lemma 15. *If $h^{add}(s) \neq \infty$, then:*

$$h^{add}(s) \leq \max_{a \in \mathcal{A}} c(a) \cdot (|\mathcal{P}^{\mathcal{O}}| + 1)^{|\mathcal{P}^{\mathcal{O}}|}$$

Proof. We can compute h^{add} with a forward fix-point computation like Bonet and Geffner (2001). Here, at most $|\mathcal{P}^{\mathcal{O}}|$ facts are updated by the cost of its achieving action (bounded by $\max_{a \in \mathcal{A}} c(a)$) and the cost of at most $|\mathcal{P}^{\mathcal{O}}|$ preconditions. This means in the first iteration the maximal cost is $\max_{a \in \mathcal{A}} c(a)$, then in the second $h^{add}(s) \leq \max_{a \in \mathcal{A}} c(a) \cdot (|\mathcal{P}^{\mathcal{O}}| + 1)$ and so on until arriving at $h^{add}(s) \leq \max_{a \in \mathcal{A}} c(a) \cdot (|\mathcal{P}^{\mathcal{O}}| + 1)^{|\mathcal{P}^{\mathcal{O}}|}$ for the last iteration (number $|\mathcal{P}^{\mathcal{O}}|$). □

For 0-cost one could bound the recursion depth instead of cost in an analogous way.

A.4 Runtime Analysis

This section provides the omitted result from Section 6.

Proposition 8. *Let $s \in S$. An upper-bound on $|E_s|$ is:*

$$O((|\Gamma| \cdot |\mathcal{A}| \cdot \max_{a \in \mathcal{A}} |pre(a)| \cdot \max_{a \in \mathcal{A}} |add(a)| \cdot h^{add}(s))^{h^{add}(s)})$$

Proof. To analyze the number of nodes, we define $nodeSize(i)$, $amNodes(i)$ and L_i. For each layer i, L_i represents the total amount of lifted atoms within nodes of layer i in the regression graph; $nodeSize(i)$ represents the maximal number of lifted atoms within each node in layer i; and $amNodes(i)$ represents the amount of nodes in layer i. We start by analyzing the node size, which is initially $nodeSize(0) = |\Gamma|$. Then in each step we add at most

$\max_{a \in \mathcal{A}} |pre(a)|$atoms, so:

$$\begin{aligned} & nodeSize(i) \\ \in\ & O(nodeSize(i-1) + \max_{a \in \mathcal{A}} |pre(a)|) \\ =\ & O(\sum_{j=1}^{i} \max_{a \in \mathcal{A}} |pre(a)| + |\Gamma|) \\ =\ & O(i \cdot \max_{a \in \mathcal{A}} |pre(a)| + |\Gamma|) \end{aligned}$$

The amount of nodes is bounded by the branching factor $\max_{a \in \mathcal{A}} |add(a)| \cdot |\mathcal{A}|$ and the amount of atoms this could replace in the previous layer, so:

$$amNodes(i+1) \in O(L_i \cdot \max_{a \in \mathcal{A}} |add(a)| \cdot |\mathcal{A}|)$$

Each layer is bounded by the amount of nodes and the according maximal node size in the last layer $h^{add}(s, g)$ is:

$$\begin{aligned} & L_{i+1} \in O(amNodes(i+1) \cdot nodeSize(i+1)) \\ =\ & O(L_i \cdot \max_{a \in \mathcal{A}} |add(a)| \cdot |\mathcal{A}| \cdot (i \cdot \max_{a \in \mathcal{A}} |pre(a)| + |\Gamma|)) \end{aligned}$$

We know that the last layer is $L_{h^{add}(s)}$. So as n^n is super-increasing. ($\sum_{i=1}^{n-1} i^i \leq \sum_{i=1}^{n-1} (n-1)^{n-1} = (n-1) \cdot (n-1)^{n-1} = (n-1)^n < n^n$) a very naive upper bound for the amount of nodes in the graph is:

$$O((|\Gamma| \cdot |\mathcal{A}| \cdot \max_{a \in \mathcal{A}} |pre(a)| \cdot \max_{a \in \mathcal{A}} |add(a)| \cdot h^{add}(s))^{h^{add}(s)})$$

□

Theorem 9. *h^{add} can be computed in polynomial time on lifted planning tasks with non-zero action cost where $h^{add}(s, \{p(\vec{o})\}) \leq k \in \mathbb{N}$ for all $p(\vec{o}) \in \Gamma$ and all $L \in E_s$ are acyclic.*

Proof. We can compute $h^{add}(s, \{p(\vec{o})\})$ separately for all $p(\vec{o}) \in \Gamma$. As h^{LRadd} runs in polynomial time w.r.t. the lifted planning task representation under these conditions (according to the bound on the number of nodes from Proposition 8, and since each set of lifted atoms can be checked for satisfiability in polynomial time if it is acyclic), this yields a polynomial time computation. □