Delete-Free Planning with Object Creation is Undecidable

Augusto B. Corrêa

University of Oxford, United Kingdom
University of Basel, Switzerland
augusto.blaascorrea@chch.ox.ac.uk

Abstract

In planning with object creation, actions might introduce new objects as part of their effect. While this makes the formalism more expressive, it also renders the plan existence problem undecidable. A natural next step is to ask whether simpler fragments and relaxations are still undecidable when powered with object creation. Probably the most popular fragment is delete-free planning, where actions can only add but never delete atoms. In this work, we show that delete-free planning with object creation is still undecidable. We do so by reducing the problem of deciding whether a given atom is reached by the chase procedure to the plan existence problem. Our result implies that heuristics based on the delete relaxation may not be immediately useful for the object creation setting. We then highlight which restrictions we can apply to make delete-free planning with object creation practical.

Introduction

In *planning with object creation*, actions can create new object as part of their effects (Hoffmann et al. 2009; Fuentetaja and de la Rosa 2016; Edelkamp, Lluch-Lafuente, and Moraru 2019; Corrêa et al. 2024). While this simplifies the modeling of planning tasks (e.g., Long and Fox 2003; Petrov and Muise 2023), it renders the plan existence problem undecidable (Hoffmann et al. 2009; Corrêa et al. 2024).

Nonetheless, there are techniques that can tackle this problem and that work well in different domains. In particular, Corrêa et al. (2024) showed that *lifted search* (cf. Corrêa and De Giacomo 2024) can be easily extended to support object creation without overhead. This leads us to the question of how to extend this to *heuristic search* algorithms, and, more specifically, on *how to compute good heuristics*.

A common family of heuristics are those computed over *delete relaxations* (Bonet and Geffner 2001; Hoffmann and Nebel 2001; Helmert and Domshlak 2009). The delete relaxation of a planning task simply ignores *delete effects* so that actions can only add new atoms to states but never remove them. The idea is to first compute this relaxation and then extract a heuristic estimate by either computing a *relaxed plan* or obtain a lower bound to the optimal relaxed plan. As solving the delete relaxation is easier than solving the original task (Bylander 1994; Erol, Nau, and Subrahmanian 1995), this provides a good framework to extract heuristics efficiently.

Moreover, many undecidable fragments of planning (Erol, Nau, and Subrahmanian 1995; Helmert 2002) become decidable when we restrict the input to *delete-free tasks* (i.e., tasks without delete effects, which is the same case obtained via delete relaxation).

But is this also the case for planning with object creation? Not so. In this paper, we show that the plan existence problem for delete-free planning tasks with object creation is undecidable. We show how to encode *Datalog$^{\pm}$ programs* (Calì et al. 2010; Calì, Gottlob, and Kifer 2013) as delete-free planning tasks with object creation. The technique builds on the classical Datalog encoding of delete-free tasks used by grounders (Helmert 2009; Corrêa et al. 2023) and by lifted planners (Corrêa et al. 2021, 2022). We then show that the problem of *atom containment* in the *chase* of a Datalog$^{\pm}$ program— which is undecidable (Beeri and Vardi 1984; Calì, Gottlob, and Kifer 2013)—is equivalent to finding a plan in the delete-free task.

Background

Throughout this paper, we use the following notation: $\mathcal{V}$ denotes a finite set of *variables*, $\mathcal{C}$ denotes a finite set of *constants*, $\mathcal{P}$ denotes finite set of *predicate symbols*, and $\mathcal{N}$ denotes a countably infinite set of *labeled nulls*.[1]

Each $p \in \mathcal{P}$ has an *arity* $ar(p)$. An *atom* is defined as $p(T_1, \ldots, T_{ar(p)})$, where $T_1, \ldots, T_{ar(p)} \in \mathcal{C} \cup \mathcal{V} \cup \mathcal{N}$ are its *terms*, denoted as $terms(p(T_1, \ldots, T_{ar(p)}))$. If $T_1, \ldots, T_{ar(p)} \subseteq \mathcal{C}$, the atom is *ground*. When the arity of an atom is not relevant, we write $p(\boldsymbol{T})$ where $\boldsymbol{T}$ represents a tuple of terms. We also write $free(p(T_1, \ldots, T_n))$ to represent the set of (free) variables in the atom. When the intended meaning is clear, we use set-theoretical notation for tuples as well (e.g., $t \in \boldsymbol{T}$, $\{t_1, t_2\} \subseteq \boldsymbol{T}$).

First-Order Languages

Let $\mathcal{L} = \langle \mathcal{V}, \mathcal{C}, \mathcal{P} \rangle$ be a *first-order language*. An *interpretation* over a first-order language $\mathcal{L}$ is a tuple $\mathcal{I} = \langle \mathcal{U}^{\mathcal{I}}, \{\mathrm{c}^{\mathcal{I}}\}_{\mathrm{c} \in \mathcal{C}}, \{p^{\mathcal{I}}\}_{p \in \mathcal{P}} \rangle$ consisting of

[1] These are used as fresh Skolem terms; they are used only in the definition of Datalog$^{\pm}$.

- a finite set $\mathcal{U}^{\mathcal{I}}$ of *objects* called the *universe*;
- for each constant $\mathrm{c} \in \mathcal{C}$, its interpretation $\mathrm{c}^{\mathcal{I}} \in \mathcal{U}^{\mathcal{I}}$.
- for each predicate symbol $p \in \mathcal{P}$, its interpretation $p^{\mathcal{I}} \subseteq (\mathcal{U}^{\mathcal{I}})^{ar(p)}$. We sometimes write $p(\mathrm{o}_1, \ldots, \mathrm{o}_{ar(p)})$ to indicate that $\langle \mathrm{o}_1, \ldots, \mathrm{o}_{ar(p)} \rangle \in \{p^{\mathcal{I}}\}$.

For the interpretation of predicates, we also define $\mathcal{P}^{\mathcal{I}} = \{p^{\mathcal{I}}\}_{p \in \mathcal{P}}$ to shorten notation. In our context, interpretations are always finite.

Delete-Free Planning with Object Creation

We use the formalism by Corrêa et al. (2024), but we restrict it to a positive STRIPS fragment (Fikes and Nilsson 1971), which is enough to prove undecidability.

A *delete-free planning task with object creation* is a tuple $\Pi^+ = \langle \mathcal{L}, \mathcal{A}, I, G \rangle$, where $\mathcal{L} = \langle \mathcal{P}, \mathcal{C}, \mathcal{V} \rangle$ is a first-order language; $\mathcal{A}$ is a finite set of *action schemas*; I is the *initial state*; G is the *goal*, defined below.

States are interpretations over $\mathcal{L}$. We assume a *fixed* interpretation of constants, where a constant $\mathrm{c} \in \mathcal{C}$ is always mapped to an object $\mathrm{o}_{\mathrm{c}} \in \mathcal{U}^s$, no matter the state s (i.e., $\mathrm{c}^s = \mathrm{o}_{\mathrm{c}}$ for all s). Therefore, we drop the interpretation of constants from our notation, and write states simply as $s = \langle \mathcal{U}^s, \{p^s\}_{p \in \mathcal{P}} \rangle$. (Later, created objects will be objects in $\mathcal{U}^s$ that are not interpreted by any constant.)

The goal G is a set of ground atoms, so, in particular, it only mentions constants.

A *delete-free action schema* $a \in \mathcal{A}$ is a pair $a = \langle \mathit{pre}(a), \mathit{add}(a) \rangle$ where each element is a set of atoms. The set of *variables* of a is a pair $\mathit{vars}(a) = \langle \mathit{params}(a), \mathit{fresh}(a) \rangle$, where

- $\mathit{params}(a) \subseteq \mathcal{V}$ is the set of *action parameters*;
- $\mathit{fresh}(a) \subseteq \mathcal{V}$ is the set of *fresh variables*;
- $\mathit{params}(a) \cap \mathit{fresh}(a) = \emptyset$.

We define $\mathit{vars}(\mathit{pre}(a))$ and $\mathit{vars}(\mathit{add}(a))$ as sets of variables, where $\mathit{vars}(\mathit{pre}(a)) \subseteq \mathit{params}(a)$, and $\mathit{vars}(\mathit{add}(a)) \subseteq \mathit{params}(a) \cup \mathit{fresh}(a)$. Intuitively, action parameters must be instantiated with objects in the universe of the current state, and fresh variables must be instantiated with new objects. The object selected to instantiate a fresh variable is a called a *fresh object*.

The set $\mathcal{C}$ of constants is exactly the set of constants that are mentioned in I, G, or in some action schema.

Given a state s and an action schema $a \in \mathcal{A}$, a *variable assignment function* $\sigma_{s,a}$ maps variables in $\mathit{params}(a) \cup \mathit{fresh}(a)$ to objects. We enforce the following two properties on $\sigma_{s,a}$: $\sigma_{s,a}(v) \in \mathcal{U}^s$ for all $v \in \mathit{params}(a)$; and $\sigma_{s,a}(v) \notin \mathcal{U}^s$ for all $v \in \mathit{fresh}(a)$. Any variable assignment function maps action parameters to objects in the state, and fresh variables to objects *not* in the state.

A *ground action* $\sigma_{s,a}(a)$ is *applicable* in s if $s \models \sigma_{s,a}(\mathit{pre}(a))$. The *successor state* $\mathit{succ}(s, \sigma_{s,a}(a))$ is defined as follows. Let $\mathit{new}(\sigma_{s,a})$ be the set of new objects introduced by $\sigma_{s,a}$ defined as $\mathit{new}(\sigma_{s,a}) = \{\sigma_{s,a}(v) \mid v \in \mathit{fresh}(a)\}$. Then $\mathit{succ}(s, \sigma_{s,a}) = \langle \mathcal{U}', \{p'\}_{p \in \mathcal{P}} \rangle$ is defined as (using $n = \mathit{ar}(p)$)

$$\mathcal{U}' = \mathcal{U}^s \cup \mathit{new}(\sigma_{s,a}),$$
$$p' = p^s \cup \{\langle \mathrm{o}_1, \ldots, \mathrm{o}_n \rangle \mid p(\mathrm{o}_1, \ldots, \mathrm{o}_n) \in \sigma_{s,a}(\mathit{add}(a))\}.$$

Therefore, the state space of our task can be interpreted as a graph over first-order interpretations.

A *plan* $\pi = \langle \sigma_{s_0,a_1}(a_1), \ldots, \sigma_{s_{n-1},a_n}(a_n) \rangle$ for Π^+ is a sequence of ground actions such that $s_0, \ldots, s_n$ are states where $s_0 = I$ and $\sigma_{s_{i-1},a_i}(a_i)$ is applicable in s_{i-1} and $s_i = \mathit{succ}(s_{i-1}, \sigma_{s_{i-1},a_i})$ for $1 \leq i \leq n$, and $s_n \models G$.

We define the following decision problem, which asks whether a plan exists for a given delete-free task with object creation:

Definition 1 (DELETEFREE-OBJCREATION-PLANEX) *Given a delete-free planning task with object creation Π^+, is there a plan for Π^+?*

Later, we show that this problem is undecidable.

Datalog$^{\pm}$ and the Chase Procedure

We base our definitions on the ones by Calì et al. (2010). We also assume basic knowledge about Datalog (Ceri, Gottlob, and Tanca 1989).

General A *homomorphism* from a set A_1 of atoms to a set A_2 of atoms is a mapping $h : \mathit{terms}(A_1) \mapsto \mathit{terms}(A_2)$ such that if $t \in \mathcal{C}$ then $h(t) = t$, and if $p(t_1, \ldots, t_n) \in A_1$ then $p(h(t_1), \ldots, h(t_n)) \in A_2$. Homomorphisms extend naturally to conjunctions of atoms.

Datalog$^{\pm}$ A *Datalog$^{\pm}$ program* is a pair $\mathcal{D}^{\pm} = \langle \mathcal{F}, \mathcal{R} \rangle$, where $\mathcal{F}$ is a finite set of ground atoms called the *facts*, and $\mathcal{R}$ is the set of *rules* of the form

$$\exists \boldsymbol{Y}\, q(\boldsymbol{T}, \boldsymbol{Y}) \leftarrow p_1(\boldsymbol{T}_1), \ldots, p_n(\boldsymbol{T}_n). \tag{1}$$

where $q, p_1, \ldots, p_n \in \mathcal{P}$, $\boldsymbol{T} \subseteq \bigcup_{i=1}^{n} \mathit{free}(\boldsymbol{T}_i)$ and $\boldsymbol{Y} \cap (\bigcup_{i=1}^{n} \mathit{free}(\boldsymbol{T}_i)) = \emptyset$. The existential quantification in the head ranges over an infinite *universal domain* $\mathcal{C} \cup \mathcal{N}$. Note that if $\boldsymbol{Y} = \emptyset$, then r is a "regular" Datalog rule. When $\boldsymbol{Y} \neq \emptyset$, r is called a *tuple-generating dependency* (tgd).

The set $\mathit{body}(r) = \{p_1(\boldsymbol{T}_1), \ldots, p_n(\boldsymbol{T}_n)\}$ is the *body* of the rule, and the singleton set $\mathit{head}(r) = \{q(\boldsymbol{T}, \boldsymbol{Y})\}$ is the atom in *head* of the rule. Given $r \in \mathcal{R}$, $\mathit{free}(r)$ denotes the set of free variables occurring in r.

For a rule r and a set of atoms $\mathcal{M}$, we say that $\mathcal{M}$ satisfies r, denoted $\mathcal{M} \models r$, if for every homomorphism h from $\mathit{body}(r)$ to $\mathcal{M}$, there is a homomorphism g from $\mathit{head}(r)$ to $\mathcal{M}$ that is *consistent* with h, i.e., $h(v) = g(v)$ for every $v \in \mathit{free}(r)$. These definitions are extended to sets of rules: given $\mathcal{R}$, we write $\mathcal{M} \models \mathcal{R}$ if $\mathcal{M} \models r$ for every $r \in \mathcal{R}$.

A set of atoms $\mathcal{M}$ is a *model* of $\mathcal{D}^{\pm} = \langle \mathcal{F}, \mathcal{R} \rangle$ if $\mathcal{F} \subseteq \mathcal{M}$ and $\mathcal{M} \models \mathcal{R}$. A model $\mathcal{M}$ is a *universal model* if it is a model and, given any other model $\mathcal{M}'$, then there is homomorphism from $\mathcal{M}$ to $\mathcal{M}'$.

Universal models are not necessarily unique (up to isomorphism) and can be infinite. Moreover, any two universal models $\mathcal{M}_1$ and $\mathcal{M}_2$ are homomorphically equivalent.

Example 1 *Consider the following Datalog$^\pm$ program $\mathcal{D}^\pm = \langle \mathcal{F}, \mathcal{R} \rangle$:*

$$\textit{next}(0, 1).$$
$$\exists Z\, \textit{next}(Y, Z) \leftarrow \textit{next}(X, Y).$$

The following model of $\mathcal{D}^\pm$ is a universal model:

$$\mathcal{M} = \{\textit{next}(0,1), \textit{next}(1,\nu_2), \textit{next}(\nu_2,\nu_3), \textit{next}(\nu_3,\nu_4)\dots\},$$

where $\nu_i \in \mathcal{N}$ for $i \geq 2$.

The Chase The *chase* procedure (Beeri and Vardi 1984) is used to repair a database with respect to a set of rules, such that the final result satisfies all the rules. The idea is to build a universal model step-by-step while satisfying all rules. We call both the procedure and its result as the chase. We restrict our discussion to the *oblivious chase* (Calì, Gottlob, and Kifer 2013).

Given a Datalog$^\pm$ program $\mathcal{D}^\pm = \langle \mathcal{F}, \mathcal{R} \rangle$, a set of atoms D, and a rule r like (1), we say that r is *applicable* to D if there exists a homomorphism h such that $h(\textit{body}(r)) \subseteq D$.

Let r be a rule that is applicable to D using a homomorphism h. The *extension* h' of h maps each existentially quantified variable $y \in \boldsymbol{Y}$ in $\textit{head}(r)$ to a fresh labeled null from $\mathcal{N}$ — i.e., a labeled null not occurring in D. The *result* of this application is a database $D' = D \cup h'(\textit{head}(r))$. We write this application as $D \overset{r,h}{\to} D'$.

A *chase sequence* $D_0 \overset{r_0,h_0}{\to} \cdots \overset{r_n,h_n}{\to} D_{n+1}$, where $D_0 = \mathcal{F}$, $r_1, \dots, r_n \in \mathcal{R}$, and $\langle r_i, h_i \rangle \neq \langle r_j, h_j \rangle$ for any $i \neq j$,[2] is a sequence of applications and results from $\mathcal{F}$.

The chase is not guaranteed to terminate: we might have an infinite chase sequence $D_0 \overset{r_0,h_0}{\to} \cdots \overset{r_i,h_i}{\to} D_{i+1} \overset{r_{i+1},h_{i+1}}{\to} \cdots$. For example, the chase runs forever in the program of Example 1. But we can still consider its results in the limit. We thus define:

$$\texttt{chase}(\mathcal{F}, \mathcal{R}) = \bigcup_{i=0}^{\infty} D_i$$

And its associated atom entailment problem:

Definition 2 (ATOMINCHASE) *Given a Datalog$^\pm$ program $\mathcal{D}^\pm = \langle \mathcal{F}, \mathcal{R} \rangle$ and an atom $q(\mathrm{c}_1, \dots \mathrm{c}_n)$ where $q \in \mathcal{P}$ and $\mathrm{c}_1, \dots, \mathrm{c}_n \in \mathcal{C}$, is $q(\mathrm{c}_1, \dots \mathrm{c}_n)$ in $\texttt{chase}(\mathcal{F}, \mathcal{R})$?*

The following theorem can be derived by the earlier results by Beeri and Vardi (1984), and was later explicitly proven by other works (e.g., Calì, Gottlob, and Kifer 2013):

Theorem 1 ATOMINCHASE *is undecidable.*

Undecidability of Delete-Free Planning with Object Creation

Next, we show that delete-free planning with object creation is undecidable. We will reduce ATOMINCHASE to DELETEFREE-OBJCREATION-PLANEX. But before diving into the main proof, we introduce a few useful lemmas.

[2]This forbids the application of a rule r with a same homomorphism h but with different extensions.

Assume that π^+ is a plan for a delete-free planning task with object creation Π^+. Assume also that π^+ contains two ground actions $\sigma_{s_1,a}(a)$ and $\sigma_{s_2,a}(a)$ such that

$$\sigma_{s_1,a}(v) = \sigma_{s_2,a}(v), \text{ for all } v \in \textit{params}(a). \tag{2}$$

Ground actions $\sigma_{s_1,a}(a)$ and $\sigma_{s_2,a}(a)$ are *redundant*. Redundant actions only differ in the objects they create. In the delete-free semantics we only need the objects created by the first applied redundant action (let's say, $\sigma_{s_1,a}(a)$), eliminating the necessity of the second redundant action ($\sigma_{s_2,a}(a)$) in our plan.

Lemma 2 *Let Π^+ be a delete-free planning task with object creation, and let π^+ be a plan for Π^+ containing two redundant actions $\sigma_{s_1,a}(a)$ and $\sigma_{s_2,a}(a)$, where $\sigma_{s_1,a}(a)$ occurs first. Then, there exists a plan $\hat{\pi}^+$ where $\sigma_{s_2,a}(a)$ does not occur and $|\hat{\pi}^+| < |\pi^+|$.*

Proof. First, assume the simple case when $\textit{fresh}(a) = \emptyset$. This implies that

$$\sigma_{s_1,a}(\textit{add}(a)) = \sigma_{s_2,a}(\textit{add}(a)).$$

As we are dealing with delete-free tasks, once we add an atom p to a state s, all atoms reached from s will contain p. Therefore, once $\sigma_{s_1,a}(a)$ is applied in π^+, applying $\sigma_{s_2,a}(a)$ does not add any new atom — they were all added by $\sigma_{s_1,a}(a)$. So $\sigma_{s_2,a}(a)$ has no impact in π^+, and simply removing $\sigma_{s_2,a}(a)$ from π^+ yields a plan $\hat{\pi}^+$.

Now, consider the case where $\textit{fresh}(a) \neq \emptyset$. This means that the add lists are different, because the fresh variables of a must always be instantiated with different objects.

Let $\{\mathrm{o}_1^1, \dots, \mathrm{o}_n^1\}$ and $\{\mathrm{o}_1^2, \dots, \mathrm{o}_n^2\}$, for $n \geq 1$, be the new objects introduced by $\sigma_{s_1,a}(a)$ and $\sigma_{s_2,a}(a)$ respectively. We claim that whenever we use an object o_i^2 in π^+, we can use o_i^1 instead, for $1 \leq i \leq n$.

When $\sigma_{s_2,a}(a)$ is applied, for any atom $p(\mathrm{o}_1, \dots, \mathrm{o}_i^2, \dots, \mathrm{o}_m)$ added by $\sigma_{s_2,a}(a)$ there is already an atom $p(\mathrm{o}_1, \dots, \mathrm{o}_i^1, \dots, \mathrm{o}_m)$ in the state s_2, which was added by $\sigma_{s_1,a}(a)$ (which was applied before by definition). So in any subsequent action $\sigma_{s',a'}(a')$, for which $p(\mathrm{o}_1, \dots, \mathrm{o}_i^2, \dots, \mathrm{o}_m) \in \sigma_{s',a'}(\textit{pre}(a'))$, the action is still applicable in s' if we replace o_i^2 with o_i^1, since we do not have negated atoms in the precondition.

We can then obtain a plan $\hat{\pi}^+$ by removing $\sigma_{s_2,a}(a)$ from π^+, and replacing every occurrence of the objects $\mathrm{o}_1^2, \dots, \mathrm{o}_n^2$ created by $\sigma_{s_2,a}(a)$ with their respective objects $\mathrm{o}_1^1, \dots, \mathrm{o}_n^1$ created by $\sigma_{s_1,a}(a)$. As just argued, the preconditions of all actions using o_i^2 are still applicable when replacing o_i^2 by o_i^1. Moreover, as the goal only mentions constants appearing in the initial state, its reachability is not affected by the removal of o_i^2. Last, as $\sigma_{s_2,a}(a)$ occurs after $\sigma_{s_1,a}(a)$ in π^+ (by assumption), the new plan is still applicable, since the first action is either $\sigma_{s_1,a}(a)$ or another action which were not modified nor removed.

As the new plan $\hat{\pi}^+$ has one action fewer than π^+, it follows directly that $|\hat{\pi}^+| < |\pi^+|$.

□

A plan without redundant actions is called a *simple plan*.

Lemma 3 *If a delete-free planning task with object creation Π^+ is solvable, then it has a simple plan.*

Proof. Given a plan π^+ for Π^+, we can remove redundant actions one by one as described in Lemma 2, until none is left. Note that as we remove actions, other actions might become redundant. However, as our first plan is finite, we only remove a finite number of actions from it. □

We are now ready to prove undecidability.

Theorem 4 DELETEFREE-OBJCREATION-PLANEX *is undecidable.*

Proof. Given a Datalog$^\pm$ program $\mathcal{D}^\pm = \langle \mathcal{F}, \mathcal{R} \rangle$ and a ground atom $q(\mathrm{c}_1, \ldots, \mathrm{c}_n)$, we reduce the problem of checking if $q(\mathrm{c}_1, \ldots, \mathrm{c}_n) \in \mathtt{chase}(\mathcal{F}, \mathcal{R})$ to a delete-free planning task with object creation Π^+.

Π^+ has the same predicate symbols as $\mathcal{D}$. The set $\mathcal{C}$ of constants in Π^+ contains all constants occurring in $\mathcal{F}$.

Our task has one action schema for each $r \in \mathcal{R}$. Given a rule r in the form

$$\exists \boldsymbol{Y}\, q(\boldsymbol{X}, \boldsymbol{Y}) \leftarrow p_1(\boldsymbol{X}_1), \ldots, p_n(\boldsymbol{X}_n).$$

we introduce an action schema a_r to Π^+ where

$$\begin{aligned} \mathit{params}(a_r) &= \boldsymbol{X} \\ \mathit{fresh}(a_r) &= \boldsymbol{Y} \\ \mathit{pre}(a_r) &= \mathit{body}(r) \\ \mathit{add}(a_r) &= \mathit{head}(r). \end{aligned}$$

In the initial state $I = \langle \mathcal{U}^I, \{\mathrm{c}^\mathcal{I}\}_{\mathrm{c}\in\mathcal{C}}, \{\mathcal{P}^I\}\rangle$, the interpretation of constants $\{\mathrm{c}^\mathcal{I}\}_{\mathrm{c}\in\mathcal{C}}$ maps each constant c to an object $\mathrm{o}_\mathrm{c} \in \mathcal{U}^I$, and the interpretation $\mathcal{P}^I$ contains $p(\mathrm{o}_{\mathrm{c}_1}, \ldots, \mathrm{o}_{\mathrm{c}_n})$ iff $p(\mathrm{c}_1, \ldots, \mathrm{c}_n) \in \mathcal{F}$. The universe $\mathcal{U}^I$ contains all objects mentioned in $\mathcal{P}^I$.

The goal G is the singleton set $\{q(\mathrm{o}_{\mathrm{c}_1}, \ldots, \mathrm{o}_{\mathrm{c}_n})\}$.

There is an one-to-one correspondence between actions and rules, and the initial state corresponds to the initial set I of facts $\mathcal{F}$. Assume action schema a corresponds to a rule r. Then there exists a ground action $\sigma_{s,a}(s)$ applicable in a given state s iff there also exists a homomorphism $h(\mathit{body}(r))$ such that r is applicable to the database $D = s$. It is easy to construct h from $\sigma_{s,a}$: let $h(X) = \sigma_{s,a}(X)$ for all $X \in \mathit{free}(r)$. Since $\mathit{free}(r) = \mathit{params}(a)$, this is well-defined. Moreover, we can compute the extension h' as follows: $h'(Y) = \sigma_{s,a}(Y)$ for all $Y \in \mathit{fresh}(a)$. This works because $\mathit{fresh}(a)$ corresponds to the existentially quantified variables in r. Thus, applying an action to a state is equivalent to an oblivious application of a rule in the chase, where the database corresponds to the state. The other way around (from applicable rules to ground actions) is analogous.

If there exists a chase sequence $D_0 \overset{r_0,h_0}{\rightarrow} \cdots \overset{r_n,h_n}{\rightarrow} D_{n+1}$ that reaches $q(\mathrm{c}_1, \ldots, \mathrm{c}_n)$ in the chase, then this sequence can be transformed into a sequence of ground actions, which corresponds to a plan, as $D_0 = \mathcal{F} = I$ and the goal of our task is to reach $q(\mathrm{c}_1, \ldots, \mathrm{c}_n)$. Each pair $\langle r_i, h_i \rangle$ can be converted into a ground action as just described above. Note that we can also convert any action to a pair $\langle r_i, h_i \rangle$, simply applying the inverse mapping.

Analogously, if there exists a plan for Π^+, then it can be transformed into a chase sequence corresponding to a derivation of $q(\mathrm{o}_{\mathrm{c}_1}, \ldots, \mathrm{o}_{\mathrm{c}_n})$. From Lemma 3 we know that if Π^+ is solvable, it has a simple plan, and we can transform any plan into a simple one. So it is sufficient to consider only simple plans, which are equivalent to chase sequences. As our initial state encodes the initial set of facts of $\mathcal{D}^\pm$, the goal atom $q(\mathrm{o}_{\mathrm{c}_1}, \ldots, \mathrm{o}_{\mathrm{c}_n})$ is only reachable in Π^+ (i.e., the task is solvable) iff $q(\mathrm{c}_1, \ldots, \mathrm{c}_n) \in \mathtt{chase}(\mathcal{F}, \mathcal{R})$. □

Discussion

Related Results In contrast to our result, planning with infinitely many constants and finite initial states becomes decidable when restricted to the delete-free case (Erol, Nau, and Subrahmanian 1995). The same happens to numeric planning (Hoffmann 2003). On the flip side, our result is closely related to *planning with function symbols* (Erol, Nau, and Subrahmanian 1995; Geffner 2000). Similar to our case, planning with function symbols is undecidable, and so is its delete-free version. It is possible to reduce from delete-free planning with function symbols to delete-free planning with object creation. However, we favor our proof because of the one-to-one correspondence between planning tasks and Datalog$^\pm$ programs, which immediately let us extend results for decidable fragments from the Datalog$^\pm$ literature (Calì et al. 2010; Calì, Gottlob, and Kifer 2013).

And Now? Our result also brings an indirect problem: computing heuristic based on relaxed plans is undecidable for tasks with object creation. To compute good heuristics, we cannot rely on delete-relaxation alone. We can use some insights from our theoretical results to help us come up with decidable heuristic functions. For example, we can further relax our problem by assuming that all created objects are homomorphic,[3] or refine this relaxation such that objects created by a same action schema are homomorphic. This brings us back into a decidable case — the total number of objects is now bounded — while still giving us a distance estimate. The idea is not far from the work by Horčík, Fišer, and Torralba (2022), but here we would restrict homomorphisms just for the fresh objects.

Another possibility is to study different decidable fragments of delete-free planning with object creation. One way is to look for cases where ATOMINCHASE is decidable, as we demonstrated that plan existence for delete-free planning tasks with object creation and atom entailment in the chase have a one-to-one correspondence. There is an extensive body of work introducing fragments of Datalog$^\pm$ for which the chase is guaranteed to terminate (Calì et al. 2010; Calì, Gottlob, and Kifer 2013). These fragments are usually based on the structure of the rules (e.g., guarded or linear rules). It would be an interesting first step to study if, when we encode our planning tasks as Datalog$^\pm$ programs (by adapting know conversions such as the one by Helmert (2009)), these programs fall into the decidable cases. If so, we can try to adapt the algorithms to compute models for Datalog$^\pm$ programs to extract heuristics.

[3] Here we refer to object homomorphism, not atom homomorphism as in the definition of Datalog$^\pm$.

Acknowledgments

Thanks to Giuseppe De Giacomo, Malte Helmert, Florian Pommerening, Travis Rivera Petit, Sasha Rubin, and Przemysław Wałega for their comments on earlier versions of this work, and to Phokion Kolaitis and Andreas Pieris for answering my questions about the chase procedure.

This paper was partially funded by the Swiss National Science Foundation (SNSF) as part of the project "Lifted and Generalized Representations for Classical Planning" (LGR-Plan).

References

Beeri, C.; and Vardi, M. Y. 1984. A Proof Procedure for Data Dependencies. *Journal of the ACM*, 31(4): 718–741.

Bonet, B.; and Geffner, H. 2001. Planning as Heuristic Search. *Artificial Intelligence*, 129(1): 5–33.

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence*, 69(1–2): 165–204.

Calì, A.; Gottlob, G.; and Kifer, M. 2013. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. *Journal of Artificial Intelligence Research*, 48: 115–174.

Calì, A.; Gottlob, G.; Lukasiewicz, T.; and Pieris, A. 2010. Datalog+/-: A Family of Languages for Ontology Querying. In de Moor, O.; Gottlob, G.; Furche, T.; and Sellers, A. J., eds., *Datalog Reloaded - First International Workshop (Datalog 2010)*, 351–368. Springer.

Ceri, S.; Gottlob, G.; and Tanca, L. 1989. What you Always Wanted to Know About Datalog (And Never Dared to Ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1): 146–166.

Corrêa, A. B.; and De Giacomo, G. 2024. Lifted Planning: Recent Advances in Planning Using First-Order Representations. In Larson, K., ed., *Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI 2024)*, 8010–8019. IJCAI.

Corrêa, A. B.; De Giacomo, G.; Helmert, M.; and Rubin, S. 2024. Planning with Object Creation. In Bernardini, S.; and Muise, C., eds., *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, 104–113. AAAI Press.

Corrêa, A. B.; Francès, G.; Pommerening, F.; and Helmert, M. 2021. Delete-Relaxation Heuristics for Lifted Classical Planning. In Goldman, R. P.; Biundo, S.; and Katz, M., eds., *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*, 94–102. AAAI Press.

Corrêa, A. B.; Hecher, M.; Helmert, M.; Longo, D. M.; Pommerening, F.; and Woltran, S. 2023. Grounding Planning Tasks Using Tree Decompositions and Iterated Solving. In Koenig, S.; Stern, R.; and Vallati, M., eds., *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)*. AAAI Press.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2022. The FF Heuristic for Lifted Classical Planning. In Honavar, V.; and Spaan, M., eds., *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2022)*, 9716–9723. AAAI Press.

Edelkamp, S.; Lluch-Lafuente, A.; and Moraru, I. 2019. Introducing Dynamic Object Creation to PDDL Planning. https://openreview.net/forum?id=rkxRj58y5N.

Erol, K.; Nau, D. S.; and Subrahmanian, V. S. 1995. Complexity, Decidability and Undecidability Results for Domain-Independent Planning. *Artificial Intelligence*, 76(1–2): 75–88.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2: 189–208.

Fuentetaja, R.; and de la Rosa, T. 2016. Compiling irrelevant objects to counters. Special case of creation planning. *AI Communications*, 29(3): 435–467.

Geffner, H. 2000. Functional Strips: A More Flexible Language for Planning and Problem Solving. In Minker, J., ed., *Logic-Based Artificial Intelligence*, volume 597 of *Kluwer International Series In Engineering And Computer Science*, chapter 9, 187–209. Dordrecht: Kluwer.

Helmert, M. 2002. Decidability and Undecidability Results for Planning with Numerical State Variables. In Ghallab, M.; Hertzberg, J.; and Traverso, P., eds., *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)*, 303–312. AAAI Press.

Helmert, M. 2009. Concise Finite-Domain Representations for PDDL Planning Tasks. *Artificial Intelligence*, 173: 503–535.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating 'Ignoring Delete Lists' to Numeric State Variables. *Journal of Artificial Intelligence Research*, 20: 291–341.

Hoffmann, J.; Bertoli, P.; Helmert, M.; and Pistore, M. 2009. Message-Based Web Service Composition, Integrity Constraints, and Planning under Uncertainty: A New Connection. *Journal of Artificial Intelligence Research*, 35: 49–117.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Horčík, R.; Fišer, D.; and Torralba, Á. 2022. Homomorphisms of Lifted Planning Tasks: The Case for Delete-free Relaxation Heuristics. In Honavar, V.; and Spaan, M., eds., *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2022)*, 9767–9775. AAAI Press.

Long, D.; and Fox, M. 2003. The 3rd International Planning Competition: Results and Analysis. *Journal of Artificial Intelligence Research*, 20: 1–59.

Petrov, A.; and Muise, C. 2023. Automated Planning Techniques for Elementary Proofs in Abstract Algebra. In *ICAPS 2023 Scheduling and Planning Applications woRKshop*.