

Planning with Dynamically Changing Domains

Mikhail Soutchanski¹, Yongmei Liu²

¹Toronto Metropolitan University (TMU), Toronto, Canada

²Sun Yat-sen University, Guangzhou, China

Abstract

In classical planning and conformant planning, it is assumed that there are finitely many named objects given in advance, and only they can participate in actions and in fluents. This is the Domain Closure Assumption (DCA). However, there are practical planning problems where the set of objects changes dynamically as actions are performed. We formulate the planning problem in first order logic, assume an initial theory is a finite consistent set of fluent literals, discuss when this guarantees that in every situation there are only finitely many possible actions, impose a finite bound on the length of the plan, and propose to organize search over sequences of actions that are grounded at run-time. We show soundness and completeness of our approach. It can be used to solve the bounded planning problems without DCA that belong to intersection of sequential generalized planning (without sensing actions) and conformant planning, restricted to the case without the disjunction over fluent literals. We discuss a proof-of-the-concept implementation of our planner.

1 Introduction

In a generalization of the British TV show Countdown (numbers round), the task is how to produce the target number out of six given integers using only standard arithmetical operations with the condition that once a number has been used, it is no longer available for subsequent calculations. This is an instance of a planning problem in the case when fluents and actions have parameters that vary over infinite domains, knowledge about an initial state is incomplete, and objects can be created or destroyed during planning. This requires planning over dynamically changing domains. A planning algorithm has to search for a correct sequential plan with respect to several different infinitely sized logical models. It turns out, the foundations for this kind of planning research have been developed years ago, as we explain next.

The real world around us can sometimes be conveniently understood in computational terms as a very large symbolic data base with incomplete knowledge, since a logical reconstruction of this open-world data base is a scalable logical theory that supports computationally tractable (sometimes) reasoning about incomplete knowledge. This perspective emerged in the early years of knowledge representation, for example, see [Reiter, 1980; Reiter, 1986; Lin and Reiter, 1997; Levesque, 1998; Liu and Levesque, 2003], and inspired significant subsequent research, for example, in description logic [Baader *et al.*, 2003]. Surprisingly, it has not been

widely explored as much as should be in automated sequential planning with deterministic actions [Ghallab *et al.*, 2004; Geffner and Bonet, 2013].

There are conceptual connections between our research and several popular directions in modern planning. We clarify these connections next before we explain our contribution.

In classical and conformant planning, it is usually assumed that there are finitely many explicitly named objects given in advance, and no objects are created or destroyed: this is the Domain Closure Assumption (DCA) [Reiter, 1980]. Moreover, in conformant planning, incomplete knowledge about an initial state is usually formulated using disjunctions of fluents, but this requires combinatorial case analysis [Hoffmann and Brafman, 2006; Palacios and Geffner, 2009; Grastien and Scala, 2020]. These papers include the case where an initial state is a finite consistent set of fluent *literals*, that is, there are fluents known to be true and fluents known to be false, but these and other papers implicitly require DCA. To the best of our knowledge, the case of planning with incomplete knowledge based on fluent literals without DCA has not been explored in conformant planning.

Moreover, in classical, conformant and numerical planning, it is common to restrict the parameters of fluents and actions to finite domains, despite that there are practical examples of the planning problems that require actions with control parameters that vary over infinite domains (such as fuel, weight); for example, see [Savas *et al.*, 2016].

In the active research area of generalized planning (GP), the planning problem has to be solved for multiple instances at once. To give a few examples, research has ranged from contingent iterative planning (with sensing) [Hu and Levesque, 2010; Srivastava, 2010; Belle and Levesque, 2016], to using abstractions [Bonet and Geffner, 2020; Illanes and McIlraith, 2019], to exploring how generalized planning can be done with existentially quantified goals [Funkquist *et al.*, 2024], or to GP as a heuristic search approach that uses a GP program to solve a set of classical planning instances [Segovia-Aguas *et al.*, 2024]. The latter paper and [Celorrio *et al.*, 2019] review previous work in GP.

We consider the generalized planning problem in first order logic, with an (existentially quantified) goal formula, as a search for a plan from a consistent set of fluent literals and additional axioms, without the DCA, and compute a sequential plan with deterministic actions, when it exists. This plan can be applied to any (infinite) model of an initial incomplete logical theory. This planning problem requires computing action preconditions at run-time. In our setting, heuristic search over state space is infeasible since there are infinitely many models (states), and actions – as well fluents – may have parameters that vary over infinite domains. Therefore, we search for a plan over sequences of instantiated actions, and formulate the

conditions that guarantee, at every planning step, there are only finitely many actions possible. If the length of a plan is bounded, then there are only finitely many sequences of possible action that can be considered. We adapt a version of a domain independent heuristic developed for the FF planner [Hoffmann and Nebel, 2001] to facilitate automated search.

Our approach can be used to solve the bounded planning problems (without DCA) that belong to intersection of sequential generalized planning (with deterministic actions) and conformant planning, restricted to the case without the disjunction over fluent literals. We discuss a proof-of-the-concept implementation of our heuristic planner, and demonstrate that it can solve a few instances without DCA.

2 Preliminaries

2.1 Proper KBs and V

We use a standard first-order (FO) language with equality, a countably infinite set of constants $\{C_1, C_2, \dots\}$, and no other function symbols. We restrict our attention to standard interpretations, where equality is identity, and there is a bijection between the set of constants and the domain of discourse. This restriction can be captured by a set of axioms \mathcal{E} , consisting of the axioms of equality and the set of formulas $\{C_i \neq C_j \mid i \neq j\}$. We use ρ to range over atoms whose arguments are distinct variables. We use e to range over ewffs, that is, quantifier-free formulas with only equalities (i.e. no predicates). We use $\forall\phi$ to denote the universal closure of ϕ . We let θ range over substitutions of all variables by constants, and write $\phi\theta$ as the result of applying θ to ϕ . We use the letter d to range over the constants. We write $\Gamma \models_{\mathcal{E}} \phi$ to denote $\mathcal{E} \cup \Gamma \models \phi$.

Intuitively, a proper KB represents a consistent set of ground literals. It generalizes databases by allowing incomplete knowledge about some of the elements.

Definition 1. A KB Γ is proper if $\mathcal{E} \cup \Gamma$ is consistent and Γ is a finite set of formulas of the form $\forall(e \supset \rho)$ or $\forall(e \supset \neg\rho)$.

Since the logical entailment problem for proper KBs is undecidable, Levesque [Levesque, 1998] proposes a reasoning scheme called V for proper KBs. Given a proper KB and a query, V returns one of three values 0 (known false), 1 (known true), or $\frac{1}{2}$ (unknown) as follows:

1. $V[\Gamma, \rho\theta] = \begin{cases} 1 & \text{if there is a } \forall(e \supset \rho) \text{ in } \Gamma \text{ s.t.} \\ & V[\Gamma, e\theta] = 1 \\ 0 & \text{if there is a } \forall(e \supset \neg\rho) \text{ in } \Gamma \text{ s.t.} \\ & V[\Gamma, e\theta] = 1 \\ \frac{1}{2} & \text{otherwise} \end{cases}$
2. $V[\Gamma, (d = d')] = 1$ if d and d' are identical constants, and 0 otherwise;
3. $V[\Gamma, \neg\phi] = 1 - V[\Gamma, \phi]$;
4. $V[\Gamma, (\phi \vee \psi)] = \max\{V[\Gamma, \phi], V[\Gamma, \psi]\}$;
5. $V[\Gamma, \exists x\phi] = \max\{V[\Gamma, \phi_d^x] \mid d \in H_1^+(\Gamma \cup \{\phi\})\}$, where $H_1^+(\Gamma \cup \{\phi\})$ includes all constants from Γ , from ϕ , and one extra constant that represents infinitely many objects not mentioned in $\Gamma \cup \{\phi\}$.

Note that if a query includes m quantifiers, then m extra constants can be required to answer the queries.

V is sound but incomplete. Levesque proposes a certain normal form called \mathcal{NF} , and shows that V is complete for queries in \mathcal{NF} :

Theorem 2. [Levesque, 1998] Suppose Γ is proper. Then for every ϕ in \mathcal{NF} , $\Gamma \models_{\mathcal{E}} \phi$ iff $V[\Gamma, \phi] = 1$; and $\Gamma \models_{\mathcal{E}} \neg\phi$ iff $V[\Gamma, \phi] = 0$.

We will not review the definition of \mathcal{NF} . The intuition behind \mathcal{NF} is that different parts of a formula must be logically independent; an example of a formula not in \mathcal{NF} is $(p \vee \neg p)$.

2.2 The Situation Calculus

The situation calculus (SC) is a logical approach to representation and reasoning about actions and their effects. It was introduced by John McCarthy, and it was refined by Reiter [Reiter, 2001] who introduced the Basic Action Theories (BAT). Unlike the notion of state that is common in model-based planning, SC relies on situation, namely a sequence of actions, which is a concise symbolic representation and a convenient proxy for the state in the cases when all actions are deterministic [Levesque *et al.*, 1998; Lin, 2008b]. We use variables s, s', s_1, s_2 for situations, variables a, a' for actions, and \bar{x}, \bar{y} for tuples of object variables. The constant S_0 represents the initial situation, and the successor function $do : action \times situation \mapsto situation$, e.g., $do(a, s)$, denotes situation that results from doing action a in previous situation s . The terms σ, σ' denote situation terms, and $A_i(\bar{x})$, or $\alpha, \alpha_1, \alpha_2, \alpha'$, represent action functions and action terms, respectively. The shorthand $do([\alpha_1, \dots, \alpha_n], S_0)$ represents situation $do(\alpha_n, do(\dots, do(\alpha_1, S_0) \dots))$ resulting from execution of actions $\alpha_1, \dots, \alpha_n$ in S_0 . The relation $\sigma \sqsubset \sigma'$ between situations terms σ and σ' means that σ is an initial sub-sequence of σ' . Any predicate symbol $F(\bar{x}, s)$ with exactly one situation argument s and possibly a tuple of object arguments \bar{x} is called a (relational) fluent. Without loss of generality, we consider only relational fluents in this paper. A first order logic (FO) formula $\psi(s)$ is called *uniform in s* if all fluents in ψ mention only situation s as their situation argument, and $\psi(s)$ has no quantifiers over situations.

The basic action theory (BAT) \mathcal{D} is the conjunction of the following classes of axioms $\mathcal{D} = \Sigma \wedge \mathcal{D}_{ss} \wedge \mathcal{D}_{ap} \wedge \mathcal{D}_{una} \wedge \mathcal{D}_{S_0}$. We use examples from the Blocks World (BW) domain [Lin and Reiter, 1997; Liu and Lakemeyer, 2009]. For brevity, all \bar{x}, a, s variables are assumed \forall -quantified at front.

\mathcal{D}_{ap} is a set of action precondition axioms of the form

$$poss(A(\bar{x}), s) \leftrightarrow \Pi_A(\bar{x}, s),$$

where $poss(a, s)$ is a new predicate symbol that means that an action a is possible in situation s , $\Pi_A(\bar{x}, s)$ is a formula uniform in s , and A is an n -ary action function. In most planning benchmarks, the formula Π_A is simply a conjunction of fluent literals and possibly negations of equality. We consider a version of BW, where there are three actions: *move-b-to-b*(x, y, z), move a block x from a block y to another block z , *move-b-to-t*(x, y), move a block x from a block y to the table, *move-t-to-b*(x, z), move a block x from the table to z .

$$\begin{aligned}
\text{poss}(\text{move-b-to-b}(x,y,z),s) &\leftrightarrow \text{clear}(x,s) \wedge \text{clear}(z,s) \wedge \\
&\quad \text{on}(x,y,s) \wedge x \neq z. \\
\text{poss}(\text{move-b-to-t}(x,y),s) &\leftrightarrow \text{clear}(x,s) \wedge \text{on}(x,y,s), \\
\text{poss}(\text{move-t-to-b}(x,z),s) &\leftrightarrow \text{ontable}(x,s) \wedge \\
&\quad \text{clear}(x,s) \wedge \text{clear}(z,s).
\end{aligned}$$

Let \mathcal{D}_{ss} be a set of successor state axioms (SSA):

$$F(\bar{x}, \text{do}(a,s)) \leftrightarrow \gamma_F^+(\bar{x}, a, s) \vee F(\bar{x}, s) \wedge \neg \gamma_F^-(\bar{x}, a, s),$$

where \bar{x} is a tuple of object arguments of the fluent F , and each of the γ_F 's is a disjunction of uniform formulas $[\exists \bar{z}]. a = A(\bar{u}) \wedge \phi(\bar{x}, \bar{z}, s)$, where $A(\bar{u})$ is an action with a tuple \bar{u} of object arguments, $\phi(\bar{x}, \bar{z}, s)$ is a context condition, and $\bar{z} \subseteq \bar{u}$ are optional object arguments; possibly $\bar{x} \subset \bar{u}$. If \bar{u} in an action function $A(\bar{u})$ does not include any z variables, then there is no optional $\exists \bar{z}$ quantifier. When $\bar{x} \subset \bar{u}$, that is the tuple of action arguments \bar{u} contains all fluent arguments \bar{x} , and possibly contains \bar{z} , we say that the action $A(\bar{u})$ has a *local effect*. A BAT is called a *local-effect* BAT if all of its actions have only local effects. In a local-effect action theory, each action can change the fluent values only for objects explicitly named as arguments of the action. If (optional) conditions $\phi(\bar{x}, \bar{z})$ do not mention s , and have no quantifiers, then SSA is called *context-free*. If there are no context conditions, then SSA is called *strictly context free* (SCF). In BW, we consider fluents $\text{clear}(x, s)$, block x has no blocks on top of it, $\text{on}(x, y, s)$, block x is on block y in situation s , $\text{ontable}(x, s)$, block x is on the table in s . The following SSAs are local-effect (with implicit outside $\forall x, \forall y, \forall a, \forall s$).

$$\begin{aligned}
&\text{clear}(x, \text{do}(a, s)) \leftrightarrow \\
&\exists y, z (a = \text{move-b-to-b}(y, x, z)) \vee \exists y (a = \text{move-b-to-t}(y, x)) \vee \\
&\quad \text{clear}(x, s) \wedge \neg \exists y, z (a = \text{move-b-to-b}(y, x, z)) \wedge \\
&\quad \neg \exists y (a = \text{move-t-to-b}(y, x)), \\
&\text{on}(x, y, \text{do}(a, s)) \leftrightarrow \\
&\exists z (a = \text{move-b-to-b}(x, z, y)) \vee \exists y' (a = \text{move-t-to-b}(x, y', y)) \vee \\
&\quad \text{on}(x, y, s) \wedge \neg \exists z (a = \text{move-b-to-b}(x, z, y)) \wedge \\
&\quad \neg \exists y' (a = \text{move-b-to-t}(x, y')), \\
&\text{ontable}(x, \text{do}(a, s)) \leftrightarrow \exists y (a = \text{move-b-to-t}(x, y)) \vee \\
&\quad \text{ontable}(x, s) \wedge \neg \exists y (a = \text{move-t-to-b}(x, y)).
\end{aligned}$$

\mathcal{D}_{una} is a finite set of unique name axioms (UNA) for actions and named objects. For example,

$$\begin{aligned}
&\text{move-b-to-b}(x, y, z) \neq \text{move-b-to-t}(x', y'), \\
&\text{move-b-to-t}(x, y) = \text{move-b-to-b}(x', y') \rightarrow x = x' \wedge y = y'.
\end{aligned}$$

\mathcal{D}_{S_0} is a set of FO formulas whose only situation term is S_0 . It specifies the initial values of fluents.

Finally, the foundational axioms Σ are generalization of axioms for a single successor function (see Section 3.1 in [Enderton, 2001]) since SC has a family of successor functions $\text{do}(\cdot, s)$, and each situation may have multiple successors (omit $\forall s_1, a_1, s_2, a_2, s, a, s'$ for brevity).

$$\text{do}(a_1, s_1) = \text{do}(a_2, s_2) \rightarrow a_1 = a_2 \wedge s_1 = s_2$$

$$\neg(s \sqsubset S_0)$$

$$s \sqsubset \text{do}(a, s') \leftrightarrow s \sqsubseteq s', \text{ where } s \sqsubseteq s' \stackrel{\text{def}}{=} (s \sqsubset s' \vee s = s')$$

$$\forall P. (P(S_0) \wedge \forall a \forall s (P(s) \rightarrow P(\text{do}(a, s)))) \rightarrow \forall s (P(s))$$

The last second order (SO) axiom limits the sort *situation* to the smallest set containing S_0 that is closed under the application of do to an action and a situation. These axioms say that the set of situations is really a tree: there are no cycles, no

merging. Since situations are finite sequences of actions, they can be implemented as lists in PROLOG, e.g., S_0 is like $[\]$, and $\text{do}(A, S)$ adds an action A to the front of a list representing S , i.e. $[A|S]$. Therefore, in PROLOG, situations satisfy Σ , and no SO reasoning is required to plan over situations.

It is often convenient to consider only executable (legal) situations: these are action histories in which it is actually possible to perform the actions one after the other.

$s \prec s' \stackrel{\text{def}}{=} s \sqsubset s' \wedge \forall a \forall s^* (s \sqsubset \text{do}(a, s^*) \sqsubseteq s' \rightarrow \text{poss}(a, s^*))$ where $s \prec s'$ means that s is an initial sub-sequence of s' and all intermediate actions are possible. Subsequently, we use the following abbreviations: $s \preceq s' \stackrel{\text{def}}{=} (s \prec s') \vee s = s'$.

Also, $\text{executable}(s) \stackrel{\text{def}}{=} S_0 \leq s$. [Reiter, 2001] formulates

$$\begin{aligned}
\text{Theorem 3. } \text{executable}(s) &\leftrightarrow ((s = S_0) \vee \\
&\quad \exists a \exists s' (s = \text{do}(a, s') \wedge \text{poss}(a, s') \wedge \text{executable}(s')))
\end{aligned}$$

Theorem 4. [Pirri and Reiter, 1999] A basic action theory $\mathcal{D} = \Sigma \wedge \mathcal{D}_{\text{ss}} \wedge \mathcal{D}_{\text{ap}} \wedge \mathcal{D}_{\text{una}} \wedge \mathcal{D}_{S_0}$ is satisfiable iff $\mathcal{D}_{\text{una}} \wedge \mathcal{D}_{S_0}$ is satisfiable.

Thus, Σ are *not* needed to check the satisfiability of \mathcal{D} .

There are two main reasoning mechanisms in SC. One of them relies on the regression operator. Another mechanism called progression is responsible for reasoning forward, where after each action α the initial theory \mathcal{D}_{S_0} is updated to a new theory \mathcal{D}_{S_α} . In this paper, we focus on progression.

Let α be a ground action, and let S_α denote the situation term $\text{do}(\alpha, S_0)$. A progression \mathcal{D}_{S_α} of \mathcal{D}_{S_0} in response to α is a set of sentences uniform in S_α such that for all queries about the future of S_α , \mathcal{D} is equivalent to $(\mathcal{D} - \mathcal{D}_{S_0}) \cup \mathcal{D}_{S_\alpha}$. In general, progression \mathcal{D}_{S_α} is SO [Lin and Reiter, 1997]. However, in a special case in the next section, Theorem 8 implies \mathcal{D}_{S_α} is in FO. Therefore, the following property of progression is a key to tractability of the queries about $\text{do}(\alpha, S_0)$.

Theorem 5. [Lin and Reiter, 1997] Let \mathcal{D}_{S_α} be a progression of the initial theory to S_α . For any sentence ϕ uniform in S_α , $\mathcal{D} \models \phi$ iff $\mathcal{D}_{S_\alpha} \models \phi$.

3 Bounded Proper Planning

In this section, we first formalize the task of bounded proper planning, then we present two examples of it.

In this paper, we consider a special form of proper KBs in the form of a finite set of ground literals, which we call a finite grounded proper (FGP) KB. Let Γ be FGP. For a predicate P , we let KP denote $\{\vec{d} \mid P(\vec{d}) \in \Gamma\}$, the set of tuples where P is known to be true and $K\neg P$ for $\{\vec{d} \mid \neg P(\vec{d}) \in \Gamma\}$, the set of tuples where P is known to be false. Then the V procedure is simplified as follows:

$$V[\Gamma, P(\vec{d})] = \begin{cases} 1 & \text{if } P(\vec{d}) \in \Gamma \\ 0 & \text{if } \neg P(\vec{d}) \in \Gamma \\ \frac{1}{2} & \text{otherwise} \end{cases}$$

Recall that conjunctive queries are a common class of formulas for retrieving information from a database, e.g., see [Chandra and Merlin, 1977; Abiteboul *et al.*, 1995]. In this paper, we extend them with safe dis-equalities, and have the following definition:

Definition 6. An extended conjunctive query (ECQ) is of the form $\exists \bar{x} \phi(\bar{x}, \bar{y})$, where ϕ is a conjunction of positive literals and safe disequalities, that is, disequalities between variables or variables and constants such that each variable has to appear in at least one positive literal.

A sufficient condition for a formula in negation normal form to be in \mathcal{NF} is that any two literals in the formula are conflict-free, that is, either they have the same polarity, or they use different predicates, or they use different constants at some argument position. Therefore, extended conjunctive queries are in \mathcal{NF} . Thus, V is sound and complete for ECQ.

[Liu and Levesque, 2005] proposes a method of progression of proper KBs for local-effect and SCF action theories. In this paper, we define a generalization of SCF action theories, called weakly context-free, and adapt their result to compute progression of finite grounded proper KBs for weakly context-free action theories.

Definition 7. A successor state axiom for fluent $F(\bar{x}, s)$ is weakly context-free (WCF) if $\gamma_F^+(\bar{x}, a)$ and $\gamma_F^-(\bar{x}, a)$ are disjunctions of formulas of the form $\exists \bar{z}[a = A(\bar{y}) \wedge \phi(\bar{u})]$, where A is an action function, \bar{y} contains \bar{z} , $\bar{u} = \bar{x} - \bar{y}$, and $\phi(\bar{u})$ is the conjunction, for each variable u in \bar{u} , of equalities $u = t(\bar{y})$, where $t(\bar{y})$ is a term using external (situation independent) functions. A BAT is WCF if each SSA is.

Here by external functions, we mean situation independent functions such as additions and multiplications.

So for an SCF SSA, the tuples for which the truth value of a fluent changes can be read from the action arguments. But for a WCF SSA, the tuples for which the truth value of a fluent changes can be determined from the action arguments by using external functions. Note that a WCF SSA is not necessarily local-effect, as each fluent argument in \bar{u} is not mentioned in $A(\bar{y})$.

Using unique name axioms and external functions, the instantiation of a WCF SSA on a ground action can be significantly simplified. Suppose that the SSA for fluent $F(\bar{x}, s)$ is WCF. Let $\alpha = A(\bar{C})$ be a ground action. Let $\mu(a) = \exists \bar{z}[a = A'(\bar{y}) \wedge \phi(\bar{u})]$ be a disjunct of $\gamma_F^*(\bar{x}, a)$, where $*$ is $+$ or $-$. If A is different from A' , then $\mu(\alpha)$ is equivalent to false. Otherwise, $\mu(\alpha)$ is equivalent to $\exists \bar{z}[\bar{y} = \bar{C} \wedge \phi(\bar{u})]$. Since \bar{y} contains \bar{z} and $\bar{u} = \bar{x} - \bar{y}$, by using external functions, $\mu(\alpha)$ is logically equivalent to $\bar{x} = \bar{d}$, where \bar{d} contains constants computed from external functions. Thus, $\gamma_F^*(\bar{x}, \alpha)$ is equivalent to a formula of the following form: $\bar{x} = \bar{d}_1 \vee \dots \vee \bar{x} = \bar{d}_n$. We will use $\gamma_F^*(\alpha)$ to denote the set $\{\bar{d}_i \mid i = 1, \dots, n\}$.

Although WCF is an extension of SCF, the progression of WCF actions can be done similarly to SCF actions except that we use external functions to supply fluent arguments.

Theorem 8. Let \mathcal{D} be weakly context-free, Γ be FGP, and α be a ground action. Then the progression of Γ wrt α , $\mathcal{P}(\Gamma, \alpha)$, remains FGP, and it can be computed as follows:

$$\begin{aligned} KF &:= KF - \gamma_F^-(\alpha) \cup \gamma_F^+(\alpha), \\ K\neg F &:= K\neg F - \gamma_F^+(\alpha) \cup \gamma_F^-(\alpha). \end{aligned}$$

Proof. Follows directly from Theorems 4 and 6 in [Liu and Levesque, 2005]. Note that the set $\gamma_F^+(\alpha)$ includes tuples for

which fluent becomes true thanks to α , $\gamma_F^-(\alpha)$ are tuples for which fluent becomes false due to α . $\mathcal{P}(\Gamma, \alpha)$ is in FO. \square

We now define the kind of BATs we consider in this paper.

Definition 9. We say a BAT is *proper* if the following holds: (1) The initial KB is a finite grounded proper KB; (2) all action preconditions axioms are quantifier-free extended conjunctive queries; (3) all SSAs are weakly context free.

We now formalize our bounded proper planning (BPP) problem. We use $Length(\sigma)$ for the number of actions in situation σ , i.e., $Length(do([\alpha_1, \dots, \alpha_N], S_0)) = N$ and $Length(S_0) = 0$.

Definition 10. A bounded proper planning (BPP) problem is a triple $P = (\mathcal{D}, G(s), N)$, where \mathcal{D} is a proper BAT, $G(s)$ is a goal formula, which is an extended conjunctive query uniform in s and without other free variables, and $N \geq 0$ is a bound. A solution to P is a ground σ with length $\leq N$ s.t.

$$\mathcal{D} \models_{\varepsilon} executable(\sigma) \wedge G(\sigma). \quad (1)$$

Below we present examples of bounded proper planning.

For many years, British TV has run a popular show Countdown that includes number round. We consider a version of Countdown that is modified for our purposes. The task is to produce the target integer number from a few given positive integers using only standard arithmetical operations. Once two numbers have been used in calculations, they are no longer available, but the new number calculated from those two can be used next. We allow any positive integer number as an initial, an intermediate, or a target number. For simplicity, consider only addition and multiplication.

We encode the modified Countdown problem using fluents $available(c, s)$, means that a counter c is available in situation s , and $value(c, n, s)$, means that a counter c holds the number n in s . Action $add(c_1, v_1, c_2, v_2)$ adds the value v_2 from the counter c_2 to the value v_1 of the counter c_1 , with the result stored in the counter c_1 , overwriting the previous value stored there, and making c_2 unavailable. Action $mult(c_1, v_1, c_2, v_2)$ represents the multiplication of values v_1, v_2 and storing the result in c_1 . In the following preconditions, the right hand sides are quantifier-free ECQ:

$$\begin{aligned} poss(add(c_1, v_1, c_2, v_2), s) &\leftrightarrow \\ &available(c_1, s) \wedge available(c_2, s) \wedge c_1 \neq c_2 \wedge \\ &value(c_1, v_1, s) \wedge value(c_2, v_2, s). \\ poss(mult(c_1, v_1, c_2, v_2), s) &\leftrightarrow \\ &available(c_1, s) \wedge available(c_2, s) \wedge c_1 \neq c_2 \wedge \\ &value(c_1, v_1, s) \wedge value(c_2, v_2, s). \end{aligned}$$

The successor-state axioms for the fluents are WCF:

$$\begin{aligned} available(c, do(a, s)) &\leftrightarrow \\ &available(c, s) \wedge \neg \exists c', x, y(a = add(c', x, c, y)) \wedge \\ &\neg \exists c', x, y(a = mult(c', x, c, y)). \\ value(c_1, v, do(a, s)) &\leftrightarrow \\ &\exists c_2, v_1, v_2(a = add(c_1, v_1, c_2, v_2) \wedge v = v_1 + v_2) \vee \\ &\exists c_2, v_1, v_2(a = mult(c_1, v_1, c_2, v_2) \wedge v = v_1 \times v_2) \vee \\ &value(c_1, v, s) \wedge \neg \exists c_2, v_2(a = add(c_1, v, c_2, v_2)) \wedge \\ &\wedge \neg \exists c_2, v_2(a = mult(c_1, v, c_2, v_2)) \end{aligned}$$

Consider several different initial theories. For clarity, instead of using the constant names such as C_{54} , we simply write 54 to represent this integer number. Similarly, we write 1 (or 2) to represent the first (the second) counter in actions.

Example 1. Let \mathcal{D}_{S_0} be the following proper theory

$$\begin{aligned} \forall c.(c=1 \vee c=2) &\rightarrow \text{available}(c, S_0) \\ \forall c, v.(c=1 \wedge v=4 \vee c=2 \wedge v=5) &\rightarrow \text{value}(c, v, S_0) \end{aligned}$$

Note that \mathcal{D}_{S_0} does not say which counters are not available and if there are any other counters and what their values might be. If the goal formula requires $\exists c(\text{value}(c, 20, \sigma))$, then this goal can be reached by performing a single action $\text{mult}(1, 4, 2, 5)$, since in every (infinite) model of \mathcal{D}_{S_0} this action is possible and leads to the goal state, regardless of the values in the other counters.

Example 2. Let \mathcal{D}_{S_0} be the following proper theory

$$\begin{aligned} \forall c.(c=1 \vee c=2) &\rightarrow \text{available}(c, S_0) \\ \forall c, v.(c=1 \wedge v=4 \vee c=2 \wedge v=5) &\rightarrow \neg \text{value}(c, v, S_0) \end{aligned}$$

If the goal formula requires $\exists c(\text{value}(c, 20, \sigma))$, then in the model \mathcal{M} of \mathcal{D}_{S_0} where the first counter holds the number 5, and the second counter holds the number 4, this goal can be reached by doing a single action $\text{mult}(1, 5, 2, 4)$. However, in another model \mathcal{M}' where both counters have say number 10, this action is not even possible. In this example, there is no action that is possible in all models of \mathcal{D}_{S_0} . In particular, actions operating with the numbers, say, in the counters 7 and 8 can be possible in some models, but not in all models.

Note that in the above examples, new objects are created via addition and multiplication, and DCA does not apply.

Example 3. The chopping tree example discussed in the papers about iterative generalized planning [Hu and Levesque, 2010; Srivastava, 2010] provides connections with that area. We use encoding from [Lin, 2008a], since it does not use sensing actions. Fluent $\text{down}(s)$ means the tree is down, $\text{size}(n, s)$ for a tree means n is the number of chops needed to bring the tree down. Action $\text{chop}(m)$ decreases the size of the tree from m to $m - 1$. The following are the action precondition axioms and SSAs:

$$\begin{aligned} \text{poss}(\text{chop}(m), s) &\leftrightarrow \text{size}(m, s) \wedge m \neq 0 \\ \text{size}(n, \text{do}(a, s)) &\leftrightarrow \exists m(a = \text{chop}(m) \wedge n = m - 1) \vee \\ &\quad \text{size}(n, s) \wedge a \neq \text{chop}(n) \\ \text{down}(\text{do}(a, s)) &\leftrightarrow a = \text{chop}(1) \wedge \text{size}(1, s) \vee \text{down}(s) \end{aligned}$$

There are infinitely many different models of \mathcal{D}_{S_0} where the tree has different sizes. Suppose that in \mathcal{D}_{S_0} we have $\forall x(x = 4 \rightarrow \text{size}(x, S_0))$. Then, every model of \mathcal{D}_{S_0} includes the fact that the initial size is 4, and a combination of other facts. It is easy to see that the sequence of 4 chop actions cuts the tree down. These actions are consecutively possible wrt all models. However, if \mathcal{D}_{S_0} is empty, i.e., it does not include any statements about fluent $\text{size}(x, S_0)$, then this subtle modification has significant consequences. Namely, there is no sequence of actions possible in all models that leads to situation σ where the goal formula $\text{down}(\sigma)$ holds.

4 Solving Bounded Proper Planning

Since proper BATs have different models that may include facts about infinitely many unnamed objects, it is not imme-

diately obvious how to search starting from \mathcal{D}_{S_0} for a plan that satisfies a goal formula wrt all models of the BAT \mathcal{D} . We propose to search over ground situations, since they represent sequences of instantiated actions. To make this search finitely feasible, we introduce an upper bound (a positive integer) on the number of actions, assuming that if an instance has a plan solving the problem, then its number of actions is less than a provided bound. At each step of planning, we have to find all possible actions, and we have to make sure that there are only finitely many ground actions that are possible wrt all models of the BAT \mathcal{D} . (We cannot choose an action that is possible wrt some models, but not possible wrt other models.) Thus, there might be at most finitely many ground situations that we can ever consider, and if a solution exists, then our search will find it. Therefore, the following Lemma 1 is important, since it guarantees that there are at most finitely many successors of each situation node. Recall a BAT mentions finitely many action functions $A(\bar{x})$.

Lemma 1. *Let \mathcal{D} be a proper BAT, σ be a ground situation such that $\mathcal{D} \models_{\mathcal{E}} \text{executable}(\sigma)$. For any action function $A(\bar{x})$, there are only finitely many tuples \vec{d} such that $\mathcal{D} \models_{\mathcal{E}} \text{poss}(A(\vec{d}), \sigma)$.*

Proof. Let $\mathcal{P}(\mathcal{D}_{S_0}, \sigma)$ be the progression of \mathcal{D}_{S_0} wrt ground actions in σ : it is a FGP theory, see Theorem 8. Let the precondition axiom for $A(\bar{x})$ be $\text{poss}(A(\bar{x}), s) \leftrightarrow \Pi_A(\bar{x}, s)$. By Theorem 5, $\mathcal{D} \models_{\mathcal{E}} \text{poss}(A(\vec{d}), \sigma)$ iff $\mathcal{P}(\mathcal{D}_{S_0}, \sigma) \models_{\mathcal{E}} \Pi_A(\vec{d}, \sigma)$. Since Π_A is a quantifier-free ECQ, that is, a conjunction of positive literals and safe dis-equalities, $\mathcal{P}(\mathcal{D}_{S_0}, \sigma)$ is equivalent to a finite set of ground literals, and the procedure $V(\cdot, \cdot)$ is complete for ECQ queries, it retrieves only finitely many \vec{d} such that $\mathcal{P}(\mathcal{D}_{S_0}, \sigma) \models_{\mathcal{E}} \Pi_A(\vec{d}, \sigma)$. \square

Note this proof does not require the DCA for objects. The fluent $\text{available}(\bar{x}, S)$ can be used to track objects destroyed by previous actions, and this fluent can occur in preconditions. Even if objects can be constructed or destroyed, at each planning step the set of named objects remains finite.

The planner has to search over executable sequences of actions. The state space remains implicit, since situations serve as symbolic proxies to states. Namely, each situation uniquely corresponds to fluent literals that comprise a state. Whenever a sequence of i ground actions results in a situation $\text{do}([\alpha_1, \dots, \alpha_i], S_0)$, to find the next action the planner must check among the actions $A_1(\bar{x}_1), \dots, A_k(\bar{x}_k)$ for which of the values of their object arguments these actions are possible. Since this computation is done at run-time, but not before search starts, our planner is lifted. Note that in proper BAT, checking if a conjunctive $G(\sigma)$ holds for a ground σ is easy.

The second key observation is that an efficient planner needs control that helps select for each situation the most promising possible action to do. This control can be provided by an A* algorithm that relies on a domain independent heuristic function inspired by [Hoffmann and Nebel, 2001].

The main advantage of this design is that the frontier stored in a priority queue consists of situations and their f -values computed as the sum of situation length (cost g) and a heuristic estimate h , i.e., for each ground σ , $f(\sigma) = g(\sigma) + h(\sigma)$.

Algorithm 1 A^* search over situation tree to find a plan

Input: (\mathcal{D}, G, N) - a bounded proper planning problem
Input: H - Heuristic function $H(\mathcal{D}, G, d, S_n, St)$
Output: S that satisfies (1) \triangleright Plan is the list of actions in S

```

1: procedure PLAN( $\mathcal{D}, G, N, H, S$ )
2:    $PriorityQueue \leftarrow \emptyset$   $\triangleright$  Initialize PQ
3:    $S_0.Val \leftarrow (N + 1)$ 
4:    $PriorityQueue.insert(S_0, S_0.Val)$ 
5:    $Init \leftarrow InitialState(\mathcal{D}_{S_0})$   $\triangleright$  Initialize state
6:   while not  $PriorityQueue.empty()$  do
7:      $S \leftarrow PriorityQueue.remove()$ 
8:      $Now \leftarrow Progress(Init, S)$   $\triangleright$  Current state
9:     if Satisfy( $Now, G$ ) then
10:      return  $S$   $\triangleright$  Found list of actions
11:    end if
12:     $Acts \leftarrow FindAllPossibleActions(Now)$ 
13:    if  $Acts == \emptyset$  then
14:      continue  $\triangleright$  No actions are possible in  $S$ 
15:    end if
16:    for  $A_i \in Acts$  do
17:       $S_n \leftarrow do(A_i, S)$   $\triangleright S_n$  is next situation
18:       $St \leftarrow Progress(Now, A_i)$   $\triangleright$  Next state
19:      if  $Length(S_n) \geq N$  then
20:        continue  $\triangleright S_n$  exceeds upper bound
21:      else  $d \leftarrow N - Length(S_n)$   $\triangleright d$  is depth bound
22:      end if  $\triangleright$  /*Each action has cost 1*/
23:       $S_n.Val \leftarrow Length(S_n) + H(\mathcal{D}, G, d, S_n, St)$ 
24:       $PriorityQueue.insert(S_n, S_n.Val)$ 
25:    end for
26:  end while
27:  return  $False$   $\triangleright$  No plan for bound  $N$ 
28: end procedure

```

In Algorithm 1, Line 7, the algorithm extracts the next most promising situation S from the frontier. Then, on Line 8, it computes progression Now of the initial state using the actions mentioned in S . On Line 9, there is a check for whether the goal formula G is satisfied in the current state Now . If it is, then S is returned as a plan. If not, then on Line 12, the algorithm finds all actions that are possible from the current state using the precondition axioms. If there are no actions possible from Now , then the algorithm proceeds to the next situation from the frontier. Otherwise, for each possible ground action A_i , it constructs the next situation $S_n = do(A_i, S)$, and if its length does not exceed the upper bound N , it computes the positive integer number d on Line 21 as $N - Length(S_n)$. This bound d is provided as an input to the heuristic function $H(\mathcal{D}, G, d, S_n, St)$ that does limited look-ahead up to depth d from St to evaluate situation S_n . On Line 24, S_n and its f -value $S_n.Val$ are inserted into the frontier, and then search continues until the algorithm finds a plan, or it explores all situations with at most N actions. The **for**-loop, Lines 16-24, makes sure that all possible successors of S are constructed, evaluated and inserted into the frontier. This is to guarantee completeness of Algorithm 1.

Theorem 11. *Algorithm 1 is sound and complete for BPP.*

Proof. By Lemma 1, there are only finitely many executable

situations with length $\leq N$. The algorithm enumerates them in the non-decreasing order of the f values. Thus all of them will be enumerated and checked for satisfaction of the goal in Line 9. By Theorem 5, $\mathcal{D} \models_{\mathcal{E}} G(do(\sigma, S_0))$ iff $\Gamma \models_{\mathcal{E}} G$, where $\Gamma = \mathcal{P}(\mathcal{D}_{S_0}, \sigma)$. By soundness and completeness of V for ECQs, $V[\Gamma, G] = 1$ iff $\Gamma \models_{\mathcal{E}} G$. So the algorithm is sound and complete. Obviously, it terminates. \square

Computing the heuristic function is done in two stages, with usual delete relaxation, i.e., negative effects of actions are ignored. (We do not include the pseudo-code of this algorithm due to lack of space, but the readers can find it in [Soutchanski and Young, 2023]. This algorithm follows [Hoffmann and Nebel, 2001].) First, a planning graph is built from the current state, layer-by-layer until the goal is satisfied or the state layer stops changing. Supporting actions are then found for the goal literals, going backwards through the graph. We skip the details of how $Reachability(\mathcal{D}, Goal, PG)$ is computed, since this is a minor improvement of the FF heuristic from Hoffmann and Nebel. The interested readers can find these details in [Soutchanski and Young, 2023]. A minor adaptation of the proof in Hoffmann and Nebel leads to

Theorem 12. *Suppose the state layer stops changing and the goal is not satisfied. Then the original planning problem is not solvable.*

5 Implementation and Experiments

We have developed a straightforward implementation of Algorithms 1 in PROLOG, following a logic programming approach from [Reiter, 2001] and [Levesque, 2012] with a few important differences. Reiter and Levesque represented the initial theory as a collection of atomic statements for fluents that are true, but since PROLOG has a built-in CWA and DCA this is too restrictive for our purposes. Our implementation of proper BATs considers fluents as terms, and collects initial proper KB literals into two lists of nodes, one node per fluent name, where each node includes a finite list of tuples. The first list of nodes represents those fluents which are initially known as true, and the second list of nodes represents those fluents which are initially known as false. Both lists can be empty for some fluents, if the initial proper KB does not include any literals for those fluents. When we compute progression, these two lists of nodes are updated accordingly to what are the effects of each action. Thus, we can work with infinite logical models by using two finite lists of tuples.

Our planner, domain files and problem instances have been loaded, compiled and run within ECLiPSe Constraint Logic Programming System, Ver7.0 #63 (x86_64_linux), released on April 24, 2022. We run it on desktop computer equipped with 11th Genuine Intel(R) Core(TM) i7-11700K CPU (3.60GHz) using only a single thread. PROLOG never exceeded 512M of memory limit allocated by default. The program has a minimal memory footprint since we only stored short situations in our priority queue. We could not compare our program with alternative implementations, since we are not aware about other planners that can solve our benchmarks.

First, we tried to solve a few instances of the **Countdown** benchmark that was implemented according to a proper BAT specified above. The program could easily solve the instances with up to 3 counters. When we experimented with different instances, the planner took less than 1 sec to find a solution with 2 actions. For the instances that involved 4 counters, the planner could find a sequence of 3 actions within 5 min. When the bound N was small, the planner could find a solution quickly, but otherwise, it was taking a long time. For the instances that had more than 4 counters, the planner could not find a solution within 10 min. To understand the issue, assume there are C counters, and notice that the first counter may have up to $2(C - 1)$ new values after doing all additions or multiplications once with the numbers in other counters. These new values can participate in operations with other elements in other counters, and so on. The number of objects destroyed or created by actions grows fast, and this increases the complexity of finding a solution.

Finally, we have developed a new benchmark **Mixers** designed to illustrate planning with a dynamically changing domain. Mixers is a significant modification of the well-known Logistics benchmark. In Mixers, there are vehicles that can move between locations. In addition, there are ingredients and compounds of different types. They can be loaded into vehicles, or they can be unloaded under the usual preconditions for these actions in Logistics. However, we added an extra condition that a vehicle must be empty to ensure that only one object can be loaded at a time. Imagine any powders or liquids as ingredients, and think about mixing them to produce a compound of a new type. Each type is a positive integer. Initially available ingredients or intermediary available compounds can be mixed according to given recipes. After mixing, the starting ingredients are no longer available, but a compound produced becomes available for subsequent actions. Each mix action has a unique ID that corresponds to a recipe specifying the name and type of a compound produced. The new type is computed according to a specific numerical function. In our PROLOG implementation, we make sure that a new compound is named with a new string computed as concatenation of the strings that name the starting ingredients. This is to make sure that new domain elements are never mentioned initially, and the names of new elements are produced at planning time. Note the objects can be created or destroyed by the mix actions, but without doing the actions, one cannot specify the objects in advance without extensive engineering of the domain. Therefore, this benchmark reflects well the purposes of planning with dynamically changing objects and without DCA. Some computational experiments conducted with instances demonstrated that Mixers is challenging. We used small values of N . We easily solved the instances with 3 recipes, two locations and 3 or 4 ingredients that required a few move actions and 1 mix actions. The plans with 4 or 5 actions completed in less than 1 sec. However, the plan that required doing two different mix actions at the same location, was computed in about 10min. Any more complex instances could not be solved within 10min limit. This benchmark is challenging because the available objects change quickly when several mix actions are required.

As a summary of our observations, we propose the hy-

pothesis that planning with dynamically changing objects is a computationally intensive process, since the planner has to deal with the increasing number of combinatorial possibilities. We consider our planner as a proof-of-the-concept implementation, and anticipate significant work is required to design an efficient program that can plan without DCA.

6 Related work

The previous research on generalized planning focused on iterative plans, or programs that support the search for a plan that works at once for a set of classical planning problems. Our work considers sequential planning without the DCA. In addition, we solve the planning problem for goals that are conjunctive queries with \exists -quantifiers over object variables, but except for [Francès and Geffner, 2016; Funkquist *et al.*, 2024], most previous work considered only conjunctions of ground fluents as goal formulas. Note since we ground actions at run-time in preconditions, checking whether resulting ground situation satisfies a goal query is easy in proper BATs.

Note that in contrast to [De Giacomo *et al.*, 2016], we do not require that the number of objects where fluent can hold is bounded for all s . Informally, boundedness of the set of named objects that may ever be considered by our planner becomes the consequence of working with a proper BAT and imposing the upper bound on the number of actions.

Lifted classical planning has been previously explored, e.g., see [Corrêa *et al.*, 2021; Ståhlberg, 2023]. Our work is different since we focus on planning without DCA.

[Soutchanski and Young, 2023] proposed a lifted deductive planner based on the situation calculus (SC), but their implementation required both DCA and CWA.

[Corrêa *et al.*, 2024] consider an extension of classical planning (the domain of objects is *finite*), with *complete knowledge*, but their semantics relies on an unusual object assignment. Our BPP is more general.

To the best of our knowledge, there are no other heuristic planners that can solve problems without the DCA given *incomplete* initial theory. All previously developed conformant planners require DCA. The planner in [Hoffmann and Brafman, 2006] was actually inspired by SC, and it does search over sequences of actions, but it works only at a propositional level. [Finzi *et al.*, 2000] does open-world planning, but they require DCA, see details in [Reiter, 2001].

7 Conclusion

To the best of our knowledge, our planner is the first heuristic planner that does not require DCA, can use actions with parameters ranging over infinite domains, works with incomplete knowledge, and can solve the planning problems with goal formulas that have \exists -quantifiers over objects. When a plan exists, it includes ground actions that are possible in every (infinite) model of a proper BAT. Future work may consider the case when identity of some objects is not known, and for this reason D_{S_0} may include \exists -quantifiers over objects; they can be replaced with Skolem constants. This case was discussed for proper KBs in the paper [De Giacomo *et al.*, 2011]. However, there are no heuristic planners that support unknown individuals represented with Skolem constants.

References

- [Abiteboul *et al.*, 1995] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. 1995.
- [Baader *et al.*, 2003] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. 2003.
- [Belle and Levesque, 2016] Vaishak Belle and Hector J. Levesque. Foundations for generalized planning in unbounded stochastic domains. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 15th Intern. Conf., KR 2016*, pages 380–389, 2016.
- [Bonet and Geffner, 2020] Blai Bonet and Hector Geffner. Qualitative numeric planning: Reductions and complexity. *J. Artif. Intell. Res.*, 69:923–961, 2020.
- [Celorrio *et al.*, 2019] Sergio Jiménez Celorrio, Javier Segovia-Aguas, and Anders Jonsson. A review of generalized planning. *Knowl. Eng. Rev.*, 34:e5, 2019.
- [Chandra and Merlin, 1977] Ashok Chandra and Philip Merlin. Optimal implementation of conjunctive queries in relational data bases. In *9th Annual ACM Symposium on Theory of Computing (STOC)*, pages 77–90, 1977.
- [Corrêa *et al.*, 2021] Augusto B. Corrêa, Guillem Francès, Florian Pommerening, and Malte Helmert. Delete-relaxation heuristics for lifted classical planning. In *Proceedings of the 31st ICAPS-2021*, pages 94–102, 2021.
- [Corrêa *et al.*, 2024] Augusto B. Corrêa, Giuseppe De Giacomo, Malte Helmert, and Sasha Rubin. Planning with object creation. In *34th International Conference on Automated Planning and Scheduling, ICAPS 2024*, pages 104–113, 2024.
- [De Giacomo *et al.*, 2011] Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. Efficient reasoning in proper knowledge bases with unknown individuals. In *22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 827–832, 2011.
- [De Giacomo *et al.*, 2016] Giuseppe De Giacomo, Yves Lespérance, and Fabio Patrizi. Bounded situation calculus action theories. *Artif. Intell.*, 237:172–203, 2016.
- [Enderton, 2001] Herbert Enderton. *A Mathematical Introduction to Logic*. Harcourt Press, 2nd edit., 2001.
- [Finzi *et al.*, 2000] Alberto Finzi, Fiora Pirri, and Ray Reiter. Open world planning in the situation calculus. In *Proceedings of the 7th AAI*, pages 754–760, 2000.
- [Francès and Geffner, 2016] Guillem Francès and Hector Geffner. \exists -strips: Existential quantification in planning and constraint satisfaction. In Subbarao Kambhampati, editor, *25th International Joint Conference on Artificial Intelligence, IJCAI-2016*, pages 3082–3088, 2016.
- [Funkquist *et al.*, 2024] Martin Funkquist, Simon Ståhlberg, and Hector Geffner. Learning to ground existentially quantified goals. In *21st Intern. Conf. on Principles of Knowledge Represent. and Reasoning*, pages 856 – 866, 2024.
- [Geffner and Bonet, 2013] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publ., 2013.
- [Ghallab *et al.*, 2004] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.
- [Grastien and Scala, 2020] Alban Grastien and Enrico Scala. CPCES: A planning framework to solve conformant planning problems through a counterexample guided refinement. *Artif. Intell.*, 284:103271, 2020.
- [Hoffmann and Brafman, 2006] Jörg Hoffmann and Ronen Brafman. Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence*, 170(6–7):507–541, 2006.
- [Hoffmann and Nebel, 2001] Jörg Hoffmann and Bernhard Nebel. The FF System: Fast Plan Generation Through Heuristic Search. *J. Artif. Intell. Res.*, 14:253–302, 2001.
- [Hu and Levesque, 2010] Yuxiao Hu and Hector Levesque. A correctness result for reasoning about one-dimensional planning problems. In *12th Intern. Conf on Principles of Knowledge Representation and Reasoning*, 2010.
- [Illanes and McIlraith, 2019] León Illanes and Sheila A. McIlraith. Generalized planning via abstraction: Arbitrary numbers of objects. In *The 33d AAAI Conference on Artificial Intelligence, AAAI 2019*, pages 7610–7618, 2019.
- [Levesque *et al.*, 1998] H.J. Levesque, F. Pirri, and R. Reiter. Foundations for the situation calculus. *Linköping Electronic Articles in Computer and Information Science*. Available at: <http://www.ep.liu.se/ea/cis/1998/018/>, vol. 3, N 18, 1998.
- [Levesque, 1998] Hector J. Levesque. A Completeness Result for Reasoning with Incomplete First-Order KBs. In *KR-1998*, pages 14–23, 1998.
- [Levesque, 2012] Hector J. Levesque. *Thinking as Computation: A First Course*. MIT Press, 2012.
- [Lin and Reiter, 1997] Fangzhen Lin and Raymond Reiter. How to Progress a Database. *Artificial Intelligence*, 92:131–167, 1997.
- [Lin, 2008a] Fangzhen Lin. Proving goal achievability. In *11th Intern Conf. Principles of Knowledge Representation and Reasoning*, pages 621–628, 2008.
- [Lin, 2008b] Fangzhen Lin. Situation calculus. In *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 649–669. Elsevier, 2008.
- [Liu and Lakemeyer, 2009] Yongmei Liu and Gerhard Lakemeyer. On First-Order Definability and Computability of Progression for Local-Effect Actions and Beyond. In *21st IJCAI-2009*, pages 860–866, 2009.
- [Liu and Levesque, 2003] Yongmei Liu and Hector J. Levesque. A tractability result for reasoning with incomplete first-order knowledge bases. In *IJCAI-03, Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 83–88, 2003.

- [Liu and Levesque, 2005] Yongmei Liu and Hector J. Levesque. Tractable reasoning with incomplete first-order knowledge in dynamic systems with context-dependent actions. In *19th International Joint Conference on Artificial Intelligence, IJCAI-2005*, pages 522–527, 2005.
- [Palacios and Geffner, 2009] Héctor Palacios and Hector Geffner. Compiling uncertainty away in conformant planning problems with bounded width. *J. Artif. Intell. Res.*, 35:623–675, 2009.
- [Pirri and Reiter, 1999] Fiora Pirri and Ray Reiter. Some contributions to the metatheory of the situation calculus. *Journal of the ACM (JACM)*, 46(3):325–361, 1999.
- [Reiter, 1980] Raymond Reiter. Equality and Domain Closure in First-Order DBs. *J. ACM*, 27(2):235–249, 1980.
- [Reiter, 1986] Raymond Reiter. A sound and sometimes complete query evaluation algorithm for relational databases with null values. *J. ACM*, 33(2):349–370, 1986.
- [Reiter, 2001] Raymond Reiter. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT, <http://cognet.mit.edu/book/knowledge-action>, 2001.
- [Savas *et al.*, 2016] Emre Savas, Maria Fox, Derek Long, and Daniele Magazzeni. Planning using actions with control parameters. In *22nd European Conference on Artificial Intelligence ECAI*, pages 1185–1193, 2016.
- [Segovia-Aguas *et al.*, 2024] Javier Segovia-Aguas, Sergio Jiménez Celorrio, and Anders Jonsson. Generalized planning as heuristic search: A new planning search-space that leverages pointers over objects. *Artif. Intell.*, 330:104097, 2024.
- [Soutchanski and Young, 2023] Mikhail Soutchanski and Ryan Young. Planning as theorem proving with heuristics. In *Proceedings of the the Italian Workshop on Planning and Scheduling, co-located with AIxIA-2023*, volume 3585 of *CEUR Workshop Proceedings*, 2023.
- [Srivastava, 2010] Siddharth Srivastava. *Foundations and Applications of Generalized Planning*. PhD thesis, Department of Computer Science, University of Massachusetts Amherst, 2010.
- [Ståhlberg, 2023] Simon Ståhlberg. Lifted successor generation by maximum clique enumeration. In *ECAI 2023 - 26th European Conference on Artificial Intelligence*, pages 2194–2201, 2023.