A Uniform Abstraction Framework for Generalized Planning

Zhenhe Cui[1] , **Yongmei Liu**[1*] , **Kailun Luo**[2]
[1]Dept. of Computer Science, Sun Yat-sen University, Guangzhou 510006, China
[2]Dongguan University of Technology, Dongguan 523808, China
cuizhh3@mail2.sysu.edu.cn, ymliu@mail.sysu.edu.cn, luokl@dgut.edu.cn

Abstract

Generalized planning aims at finding a general solution for a set of similar planning problems. Abstractions are widely used to solve such problems. However, the connections among these abstraction works remain vague. Thus, to facilitate a deep understanding and further exploration of abstraction approaches for generalized planning, it is important to develop a uniform abstraction framework for generalized planning. Recently, Banihashemi et al. proposed an agent abstraction framework based on the situation calculus. However, expressiveness of such an abstraction framework is limited. In this paper, by extending their abstraction framework, we propose a uniform abstraction framework for generalized planning. We formalize a generalized planning problem as a triple of a basic action theory, a trajectory constraint, and a goal. Then we define the concepts of sound abstractions of a generalized planning problem. We show that solutions to a generalized planning problem are nicely related to those of its sound abstractions. We also define and analyze the dual notion of complete abstractions. Finally, we review some important abstraction works for generalized planning and show that they can be formalized in our framework.

1 Introduction

Generalized planning, where a single plan works for possibly infinitely many similar planning problems, has received continual attention in the AI community [Srivastava *et al.*, 2008; Hu and De Giacomo, 2011; Srivastava *et al.*, 2011; Segovia-Aguas *et al.*, 2016; Bonet and Geffner, 2018; Illanes and McIlraith, 2019]. For example, the generalized plan "while the block A is not clear, pick the top block above A and place it on the table" meets the goal $clear(A)$ no matter how many blocks the tower has.

Abstractions are widely used to solve generalized planning problems. The idea is to develop an abstract model of the problem that suppresses less important details, find a solution in the abstract model, and use this solution to guide the search for a solution in the concrete model. For example, Srivastava *et al.* [2011] introduced *qualitative numerical planning* (QNP) problems, which represents a set of quantitative numerical planning problems. They showed that a QNP problem P can be easily abstracted into a FOND (*fully observable non-deterministic planning*) problem whose strong cyclic solutions that terminate in P are solutions to P. To explain why the termination condition is needed and how it can be removed, Bonet *et al.* [2017] showed that a planning problem can be extended with a trajectory constraint and the solutions to the problem are programs whose executions satisfying the constraint terminate. Bonet and Geffner [2018] considered solving generalized classical planning problems. They showed that if such a problem P can be abstracted into a QNP problem P' and if the abstraction is sound, then the solution of P' is a solution of P. Illanes and McIlraith [2019] considered solving a class of generalized classical planning problems by automatically deriving a sound QNP abstraction from an instance of the problem, by introducing a counter for each set of indistinguishable objects.

The existing abstraction works for generalized planning are closely related to each other. However, the similarities and differences of them remain vague. Thus, to facilitate a deep understanding and further exploration of abstraction approaches for generalized planning, it is important to develop a uniform theoretical framework for them.

Recently, Banihashemi *et al.* [2017] proposed an agent abstraction framework based on the situation calculus [Reiter, 2001] and the Golog [Levesque *et al.*, 1997] agent programming language. They relate a high-level action theory to a low-level action theory by the notion of a refinement mapping, which specifies how each high-level action is implemented by a low-level Golog program and how each high-level fluent can be translated into a low-level formula. They define the concepts of sound/complete abstractions of a low-level action theory and prove properties that relate the behavior of a low-level action theory to the behavior of its sound/complete abstractions. However, expressiveness of such an abstraction framework is limited, and cannot serve as a uniform abstraction framework for generalized planning.

In this paper, by extending the abstraction framework of Banihashemi *et al.*, we propose a uniform abstraction framework for generalized planning. We first extend the situation calculus with counting and use non-deterministic Golog

*Corresponding Author

programs to represent actions with non-deterministic effects, and formalize a generalized planning problem as a triple of a basic action theory, a trajectory constraint, and a goal. Then we define the concepts of sound/complete abstractions of a generalized planning problem. We show that solutions to a generalized planning problem are nicely related to those of its sound/complete abstractions. Finally, we review some important abstraction works for generalized planning and show that they can be formalized in our framework.

2 Preliminaries

In this section, we introduce the situation calculus extended with infinite histories and $Golog$.

The situation calculus [Reiter, 2001] is a many-sorted first-order language with some second-order ingredients suitable for describing dynamic worlds. There are three disjoint sorts: $action$ for actions, $situation$ for situations, and $object$ for everything else. The language also has the following components: a situation constant S_0 denoting the initial situation; a binary function $do(a, s)$ denoting the successor situation to s resulting from performing action a; a binary relation $Poss(a, s)$ indicating that action a is possible in situation s; a binary relation $s \sqsubseteq s'$, meaning situation s is a sub-history of s'; a set of relational (functional) fluents, *i.e.*, predicates (functions) taking a situation term as their last argument. A formula is uniform in s if it does not mention any situation term other than s. We call a formula with all situation arguments eliminated a situation-suppressed formula. For a situation-suppressed formula ϕ, we use $\phi[s]$ to denote the formula obtained from ϕ by restoring s as the situation arguments to all fluents.

In the situation calculus, a particular domain of application can be specified by a basic action theory (BAT) of the form

$$\mathcal{D} = \Sigma \cup \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}, \text{ where}$$

1. Σ is the set of the foundational axioms for situations;
2. $\mathcal{D}_{ap}$ is a set of action precondition axioms, one for each action function A of the form $Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$, where $\Pi_A(\vec{x}, s)$ is uniform in s;
3. $\mathcal{D}_{ss}$ is a set of successor state axioms, one for each relation fluent symbol P of the form $P(\vec{x}, do(a, s)) \equiv \Phi_P(\vec{x}, a, s)$, and one for each functional fluent symbol f of the form $f(\vec{x}, do(a, s)) = y \equiv \phi_f(\vec{x}, y, a, s)$, where $\Phi_P(\vec{x}, a, s)$ and $\phi_f(\vec{x}, y, a, s)$ are uniform in s;
4. $\mathcal{D}_{una}$ is the set of unique name axioms for actions;
5. $\mathcal{D}_{S_0}$ is the initial knowledge base stating facts about S_0.

The situation calculus cannot be used to express and reason about infinite sequences of actions. Schulte and Delgrande [2004] extended the situation calculus with infinite histories. They introduce: a sort $infhist$ for infinite sequences of actions, with variables h, h'; the extended binary relation $\sqsubseteq$, where $s \sqsubseteq h$ means s is a subhistory of h; predicates $possible(s)$ and $possible(h)$ stating that no impossible action occurs in s and h respectively. The set Σ^∞ of axioms that characterize infinite histories are as follows:

- $S_0 \sqsubset h; \quad possible(S_0);$
- $s \sqsubset h \rightarrow \exists s'.s \sqsubset s' \wedge s' \sqsubset h;$
- $(s' \sqsubset s \wedge s \sqsubset h) \rightarrow s' \sqsubset h;$
- $possible(h) \equiv \forall s \sqsubset h.possible(s);$
- $possible(do(a, s)) \equiv possible(s) \wedge Poss(a, s).$

Levesque *et al.* [1997] introduced a high-level programming language Golog with the following syntax:

$$\delta ::= \alpha | \ \varphi? \mid \delta_1; \delta_2 \mid \delta_1 | \delta_2 \mid \pi x.\delta \mid \delta^*,$$

where α is an action term; φ is a situation-suppressed formula and $\varphi?$ tests whether φ holds; program $\delta_1; \delta_2$ represents the sequential execution of δ_1 and δ_2; program $\delta_1 | \delta_2$ denotes the non-deterministic choice between δ_1 and δ_2; program $\pi x.\delta$ denotes the non-deterministic choice of a value for parameter x in δ; program δ^* means executing program δ for a non-deterministic number of times.

Golog has two kinds of semantics: transition semantics and evaluation semantics. In transition semantics, a configuration of a Golog program is a pair (δ, s) of a situation s and a program δ remaining to be executed. The predicate $Trans(\delta, s, \delta', s')$ means that there is a transition from configuration (δ, s) to (δ', s') in one elementary step. The predicate $Final(\delta, s)$ means that the configuration (δ, s) is a final one, which holds if program δ may legally terminate in situation s. Please refer to [De Giacomo *et al.*, 2000] for details of the definitions of $Trans$ and $Final$. We use $\mathcal{C}$ to denote the axioms for defining $Trans$ and $Final$. In evaluation semantics, the predicate $Do(\delta, s, s')$ means that executing the program δ in situation s will terminate in a situation s'. Do can be defined with $Trans$ and $Final$ as follows:

$$Do(\delta, s, s') \doteq \exists \delta'.Trans^*(\delta, s, \delta', s') \wedge Final(\delta', s'), \text{ where}$$

$Trans^*$ denotes the reflexive transitive closure of $Trans$.

3 Extension of the Situation Calculus

To represent the property of termination, we use the situation calculus with infinite histories. To represent planning with non-deterministic actions, following [Bacchus *et al.*, 1995], we treat a non-deterministic action as a non-deterministic program. To represent numerical planning based on counting, we extend the situation calculus with counting.

The counting ability of first-order logic is very limited. Kuske and Schweikardt [2017] extended FOL by counting, getting a new logic FOCN. The key construct of FOCN are counting terms of the form $\#\bar{y}.\varphi$, meaning the number of tuples $\bar{y}$ satisfying formula φ. Formulas of FOCN are interpreted over finite structures.

To extend the situation calculus with counting, as in [Li and Liu, 2020], we make the assumption that there are finitely many non-number objects. We introduce a sort nat for natural numbers with the $+$ and $\cdot$ operations, and a function symbol μ of sort $object \rightarrow nat$. The intended interpretation of μ is a coding of objects into natural numbers. We use n, m for variables of nat, and x, y for variables of non-number objects. If $\varphi(\bar{y})$ is a formula, then $\#\bar{y}.\varphi$ is a nat term, with the same meaning as in FOCN.

Transitive closure is often used to define counting terms. Following transitive closure logic [Immerman and Vardi,

1997], we introduce the notation $[TC_{\bar{x},\bar{y}}\varphi](\bar{u},\bar{v})$, where $\varphi(\bar{x},\bar{y})$ is a formula with $2k$ free variables, $\bar{u}$ and $\bar{v}$ are two k-tuples of terms, which says that the pair $(\bar{u},\bar{v})$ is contained in the reflexive transitive closure of the binary relation on k-tuples that is defined by φ. It is defined as an abbreviation in the situation calculus using a formula in second-order logic. We omit the definition here. In case $P(x,y)$ is a binary predicate, we simply write $P^*(x,y)$ to mean $[TC_{x,y}P(x,y)](x,y)$, and write $P^+(x,y)$ to mean $P^*(x,y) \wedge x \neq y$.

To formalize planning problems where actions may have non-deterministic effects, following [Bacchus *et al.*, 1995], we represent a non-deterministic action $A(\vec{x})$ as follows: we introduce associated deterministic actions $A_d(\vec{x},\vec{y})$, and define $A(\vec{x})$ as a non-deterministic program of $A_d(\vec{x},\vec{y})$ actions. For example, suppose we have a functional fluent $f(x,s)$ and an action $dec(x)$ which decreases the value of $f(x,s)$ by an arbitrary positive amount. To represent $dec(x)$, we introduce a deterministic action $dec_d(x,y)$ which is possible when $y > 0$ and has the effect of decreasing the value of $f(x,s)$ by y. Then we have the definition $dec(x) \doteq \pi y.dec_d(x,y)$.

A particular domain of application where there are finitely many non-number objects is specified by a finite model basic action theory of the following form:

$$\mathcal{D} = \mathcal{D}_u \cup \Sigma^\infty \cup \mathcal{P} \cup \mathcal{C} \cup \mathcal{F} \cup \mathcal{D}_{def}, \text{where}$$

1. $\mathcal{D}_u = \Sigma \cup \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ as before, where $\mathcal{D}_{ap}$ and $\mathcal{D}_{ss}$ specify the action precondition and successor state axioms for the *deterministic* actions, respectively.
2. Σ^∞ is the set of axioms for infinite histories.
3. $\mathcal{P}$: the second-order axiomatization of Peano arithmetic.
4. $\mathcal{C}$ is the set of axioms for defining $Trans$ and $Final$.
5. $\mathcal{F}$ is the set of the following axioms:
   - $\forall x,y.\mu(x) = \mu(y) \rightarrow x = y$;
   - $\exists n \forall x.\mu(x) \leq n$.

   The above axioms state that the codings of different objects are different and there is a largest coding.
6. $\mathcal{D}_{def}$ is a set of definition axioms where for each *non-determinstic* action $A(\vec{x})$, there are two axioms of the form $Trans(A(\vec{x}),s,\delta,s') \equiv Trans(\delta_A(\vec{x}),s,\delta,s')$ and $Final(A(\vec{x}),s) \equiv Final(\delta_A(\vec{x}),s)$, where $\delta_A(\vec{x})$ is the definition for $A(\vec{x})$.

Note that we treat non-deterministic actions as abbreviations for non-deterministic programs, thus we do not define their preconditions and successor state axioms.

Example 1. An agent needs to clear all the blocks above a block. Her behavior is constrained by the battery level. She needs to charge her battery when its level is < 10 (we assume that the battery level is an integer between 0 and 100). Action execution may consume electricity. There are 4 relational fluents: $on(x,y,s)$, $holding(x,s)$, $clear(x,s)$, $handempty(s)$, and a functional fluent $battery\text{-}level(s)$. We also have 3 actions: $unstack(x,y)$, $drop(x)$ and $charge$. The effect of $unstack(x,y)$ is unstacking x from y, non-deterministically decreasing the battery level by 1 or 2, depending on the weight of x. To represent $unstack(x,y)$, we introduce a deterministic action $unstack(x,y,\eta)$, where η is the amount of electricity consumption. Below are example axioms from the BAT for this domain.

Precondition Axioms:

$Poss(unstack(x,y,\eta),s) \equiv on(x,y,s) \wedge 1 \leq \eta \leq 2 \wedge$
$clear(x,s) \wedge handempty(s) \wedge battery\text{-}level(s) \geq 10$

Successor State Axioms:

$clear(x,do(a,s)) \equiv (\exists y,\eta)a = unstack(y,x,\eta) \vee$
$clear(x,s) \wedge \forall z,\eta.a \neq unstack(x,z,\eta)$
$battery\text{-}level(do(a,s)) = r \equiv r = 100 \wedge a = charge \vee$
$r = battery\text{-}level(s) - 1 \wedge (\exists x)a = drop(x) \vee$
$\exists x,y,\eta.a = unstack(x,y,\eta) \wedge r = battery\text{-}level(s) - \eta.$

Non-deterministic Action Definitions:

$unstack(x,y) \doteq \pi\eta.\ unstack(x,y,\eta)$

Initial Situation Axiom:

$handempty(S_0) \wedge battery\text{-}level(S_0) \geq 10$

4 Our Abstraction Framework

In this section, based on the agent abstraction framework in [Banihashemi *et al.*, 2017], we propose a uniform abstraction framework for generalized planning.

4.1 Generalized Planning Problems and Solutions

A generalized planning (g-planning) problem can be defined as a pair of an action theory and a goal. However, in the presence of non-deterministic actions, solutions to planning problems are programs whose execution under certain trajectory constraints are guaranteed to terminate and achieve the goal. For example, strong cyclic solutions to FOND problems are policies that terminate and achieve the goal under the fairness assumption, which states that if an action occurs infinitely often, then any of its possible effects occurs infinitely often. Thus when defining g-planning problems, we include an extra component of a trajectory constraint, which is a situation calculus formula with a free variable h of infinite histories.

Definition 1. A generalized planning problem is a tuple $\mathcal{G} = \langle \mathcal{D}, C, G \rangle$, where $\mathcal{D}$ is a BAT, C is a trajectory constraint, i.e., a situation calculus formula with a free variable of infinite histories, and G, a situation-suppressed formula, is a goal condition.

For example, the fairness assumption can be expressed as follows. We use $\mathcal{ND}$ to denote the set of non-deterministic actions. For each $A \in \mathcal{ND}$, we assume that its definition is $A(\vec{x}) \doteq \pi\vec{y}.A_d(\vec{x},\vec{y})$. Then the fairness assumption is expressed as $\bigwedge_{A \in \mathcal{ND}} \forall\vec{x}.\psi_1 \supset \psi_2$, where

$$\psi_1 = \forall s \sqsubset h \exists s' \sqsubset h \exists\vec{y}.s \sqsubseteq s' \wedge do(A_d(\vec{x},\vec{y}),s') \sqsubset h,$$

$$\psi_2 = \forall\vec{y} \forall s \sqsubset h \exists s' \sqsubset h.s \sqsubseteq s' \wedge do(A_d(\vec{x},\vec{y}),s') \sqsubset h.$$

Example 1 cont'd. Let $\mathcal{D}_l$ denote the BAT specified earlier. The low-level g-planning problem is $\mathcal{G}_l = \langle \mathcal{D}_l, C_l, G_l \rangle$, where C_l is $\top$, and G_l is $clear(A)$.

Solutions to g-planning problems take the form of Golog programs. To define solutions, we introduce three abbreviations. Achieve(δ, G) means starting in S_0, there is an execution of δ that makes δ hold:

$$\text{Achieve}(\delta, G) \doteq \exists s. Do(\delta, S_0, s) \wedge G[s].$$

$\text{Ensure}(\delta, G)$ means starting in S_0, whenever program δ terminates, G is satisfied:

$$\text{Ensure}(\delta, G) \doteq \forall s. Do(\delta, S_0, s) \supset G[s].$$

$\text{Term}(\delta, s, C)$ means starting in situation s, program δ terminates under constraint C, i.e., there is no infinite execution of δ starting in s and satisfying C:

$$\text{Term}(\delta, s, C) \doteq \neg\exists h. C(h) \wedge \forall s' \sqsubset h \exists \delta' \, Trans^*(\delta, s, \delta', s').$$

Note that when $\exists s. Do(\delta, S_0, s)$ is false, $\text{Ensure}(\delta, G)$ holds trivially. The execution of a Golog program may abort when a test falsifies or the precondition of a primitive action does not hold. For example, the execution of $\phi?; \alpha$ may abort when ϕ falsifies or the precondition of α does not hold.

Definition 2. Let $\mathcal{G} = \langle \mathcal{D}, C, G \rangle$ be a g-planning problem, and δ a Golog program.

1. We say δ is a weak solution to $\mathcal{G}$ if $\mathcal{D} \models \text{Achieve}(\delta, G)$.
2. We say δ is a strong solution to $\mathcal{G}$ if

$$\mathcal{D} \models \text{Term}(\delta, S_0, C) \wedge \text{Ensure}(\delta, G) \wedge \exists s. Do(\delta, S_0, s).$$

So a weak solution is a program that may achieve the goal, but is not guaranteed to do so. A strong solution is a weak solution that, under the trajectory constraint, is guaranteed to achieve the goal unless the program execution aborts.

Example 2. There is a block on the table and the goal is to have the block at hand. We have an action *pickup*, which may succeed or fail. A weak solution is to pickup the block. Under the fairness assumption, a strong solution is to pickup the block until it is at hand. The only infinite executions of the program are unfair ones.

4.2 Simulations and Back-simulations

The concept of refinement mappings is the same as that in [Banihashemi *et al.*, 2017] except that a refinement mapping can also map a high-level function to a low-level term.

Definition 3. A function m is a refinement mapping from $\mathcal{G}_h = \langle \mathcal{D}_h, C_h, G_h \rangle$ to $\mathcal{G}_l = \langle \mathcal{D}_l, C_l, G_l \rangle$ if for each high-level deterministic or non-deterministic action type A, $m(A(\vec{x})) = \delta_A(\vec{x})$, where $\delta_A(\vec{x})$ is a low-level program; for each high-level relational fluent P, $m(P(\vec{x})) = \phi_P(\vec{x})$, where $\phi_P(\vec{x})$ is a low-level situation-suppressed formula; for each high-level functional fluent f, $m(f(\vec{x})) = \tau_f(\vec{x})$, where $\tau_f(\vec{x})$ is a low-level term, possibly a counting term.

In the following, we will relate models of $\mathcal{D}_h$ and $\mathcal{D}_l$. When we relate a model M_h of $\mathcal{D}_h$ and a model M_l of $\mathcal{D}_l$, we do not require that the two models have the same object domain, which is the case in [Banihashemi *et al.*, 2017]. By $m(A(\vec{x})) = \delta_A(\vec{x})$, we mean that the arguments of δ_A are included in $\vec{x}$, but some x_i's may not appear as arguments of δ_A. If some x_i is an argument of δ_A, it represents an object belonging to both the high-level and low-level models.

Let δ be a high-level program, and ϕ a high-level formula. We use $m(\delta)$ to denote the program resulting from replacing each high-level symbol in δ with its low-level definitions. $m(\phi)$ is similarly defined.

Example 1 cont'd. We have two high-level actions: $clearablock(x)$ and $charge$, where $clearablock(x)$ means removing the top block of the tower of x. We have two high-level functional fluents: $n(x, s)$ and $battery\text{-}level(s)$, where $n(x, s)$ counts the number of blocks above x. The initial KB is $battery\text{-}level(S_0) \geq 10$. We omit axioms from the high-level action theory $\mathcal{D}_h$. The high-level g-planning problem is $\mathcal{G}_h = \langle \mathcal{D}_h, C_h, G_h \rangle$, where C_h is also $\top$, and G_h is $n(A) = 0$. Below is the refinement mapping from $\mathcal{D}_h$ to $\mathcal{D}_l$:

$$m(clearablock(x)) =$$
$$\pi y, z. (on(y, z) \wedge on^*(z, x))?; unstack(y, z); drop(y),$$
$$m(n(x)) = \#y. \, on^+(y, x),$$

where $on^*(z, x)$ and $on^+(y, x)$ are transitive closure formulas which we introduce in Section 3.

To relate high-level and low-level models, we first define the m-isomorphism relations between high-level and low-level situations. The main difference from the definition in [Banihashemi *et al.*, 2017] is that we do not require that M_h and M_l have the same object domain, as stated earlier.

We use the following notation. For a variable assignment v, $v[s/s_h]$ denotes the assignment which maps variable s to situation s_h and is the same as v elsewhere. For a model M of the situation calculus, let Δ_S^M stand for the domain of situations in M, and S_0^M for the denotation of S_0 in M.

Definition 4. Given a refinement mapping m, a situation s_h of a high-level model M_h is m-isomorphic to a situation s_l in a low-level model M_l, written $s_h \sim_m^{M_h, M_l} s_l$, if: for any high-level relational fluent P, variable assignment v, we have $M_h, v[s/s_h] \models P(\vec{x}, s)$ iff $M_l, v[s/s_l] \models m(P)(\vec{x}, s)$; for any high-level functional fluent f, variable assignment v, we have $M_h, v[s/s_h] \models f(\vec{x}, s) = y$ iff $M_l, v[s/s_l] \models m(f)(\vec{x}, s) = y$.

Note that as before, by writing $m(P)(\vec{x}, s)$, we mean that the object arguments of $m(P)$ are included in $\vec{x}$, but some x_i's may not appear as arguments of $m(P)$. If some x_i is an argument of $m(P)$, it represents an object belonging to both M_h and M_l.

Example 1 cont'd. Let M_l be a model of $\mathcal{D}_l$ where there are 3 blocks A, B, C, and let M_h be a model of $\mathcal{D}_h$ with only one block A. Then a low-level situation where C is on B, B is on A, and $battery\text{-}level = 10$ is m-isomorphic to a high-level situation where $n(A) = 2$ and $battery\text{-}level = 10$.

Proposition 1. *Suppose $s_h \sim_m^{M_h, M_l} s_l$. Let ϕ be a high-level situation-suppressed formula. Then $M_h, v[s/s_h] \models \phi[s]$ iff $M_l, v[s/s_l] \models m(\phi)[s]$.*

In [Banihashemi *et al.*, 2017], high-level and low-level models are related by an m-bisimulation relation. To define sound/complete abstractions, we do not require the strong condition of bisimulation, so we break the bisimulation relation into the simulation and back-simulation relations. Intuitively, simulation means: whenever a refinement of a high-level action can occur, so can the high-level action, and back-simulation means the other direction. Two extra conditions distinguish our definitions from that of [Banihashemi *et al.*, 2017]: for each high level action $A(\vec{o})$, program $m(A(\vec{o}))$

terminates; we relate the infinite histories of the high-level and low-level models satisfying the trajectory constraints.

Definition 5. (m-simulation) A relation $B \subseteq \Delta_S^{M_h} \times \Delta_S^{M_l}$ is an m-simulation relation between M_h and M_l, if $\langle S_0^{M_h}, S_0^{M_l}\rangle \in B$ and the following hold:

1. $\langle s_h, s_l\rangle \in B$ implies that: $s_h \sim_m^{M_h,M_l} s_l$; for any high-level action A, and variable assignment v, $M_l, v[s/s_l] \models \text{Term}(m(A(\vec{x})), s, C_l)$, and if there is a situation s_l' s.t. $M_l, v[s/s_l, s'/s_l'] \models Do(m(A(\vec{x})), s, s')$, then there is a situation s_h' s.t. $M_h, v[s/s_h, s'/s_h'] \models Do(A(\vec{x}), s, s')$ and $\langle s_h', s_l'\rangle \in B$.
2. For any infinite high-level action sequence σ, if there is an infinite history in M_l generated by $m(\sigma)$ and satisfying C_l, then there is an infinite history in M_h generated by σ and satisfying C_h.

By an infinite history generated by a program, we mean an infinite execution of the program.

Definition 6. (m-back-simulation) A relation $B \subseteq \Delta_S^{M_h} \times \Delta_S^{M_l}$ is an m-back-simulation relation between M_h and M_l, if $\langle S_0^{M_h}, S_0^{M_l}\rangle \in B$, and the following hold:

1. $\langle s_h, s_l\rangle \in B$ implies that: $s_h \sim_m^{M_h,M_l} s_l$; for any high-level action A, and variable assignment v, $M_l, v[s/s_l] \models \text{Term}(m(A(\vec{x})), s, C_l)$, and if there is a situation s_h' s.t. $M_h, v[s/s_h, s'/s_h'] \models Do(A(\vec{x}), s, s')$, then there is a situation s_l' s.t. $M_l, v[s/s_l, s'/s_l'] \models Do(m(A(\vec{x})), s, s')$ and $\langle s_h', s_l'\rangle \in B$;
2. For any infinite high-level action sequence σ, if there is an infinite history in M_h generated by σ and satisfying C_h, then there is an infinite history in M_l generated by $m(\sigma)$ and satisfying C_l.

4.3 Sound/Complete Abstractions on Model Level

We first define sound/complete abstractions on model level, then we define sound/complete abstractions on theory level. For both model and theory levels, sound abstractions mean that high-level behavior entails low-level behavior, and complete abstractions mean the other direction.

Definition 7. We say that M_h is a *sound m-abstraction* of M_l, written $M_h \sim_m^{\rightarrow} M_l$, if there is an m-back-simulation relation B between M_h and M_l.

The following proposition generalizes the back-simulation from actions to programs.

Proposition 2. *Suppose that M_h is a sound m-abstraction of M_l via the m-back-simulation relation B. Let δ be a high-level Golog program, and $\langle s_h, s_l\rangle \in B$. Then*

1. *if there is a situation s_h' in M_h s.t. $M_h, v[s/s_h, s'/s_h'] \models Do(\delta, s, s')$, then there is a situation s_l' in M_l s.t. $M_l, v[s/s_l, s'/s_l'] \models Do(m(\delta), s, s')$ and $\langle s_h', s_l'\rangle \in B$;*
2. *if there is an infinite history in M_h generated by δ and satisfying C_h, then there is an infinite history in M_l generated by $m(\delta)$ and satisfying C_l.*

Proof. To prove 1, use structural induction on δ. For the case of tests, use Prop. 1. To prove 2, use the condition that for each high level action $A(\vec{o})$, $m(A(\vec{o}))$ terminates. □

Definition 8. We say that M_h is a *complete m-abstraction* of M_l, written $M_h \sim_m^{\leftarrow} M_l$, if there is a m-simulation relation B between M_h and M_l.

Proposition 3. *Suppose that M_h is a complete m-abstraction of M_l via the m-simulation relation B. Let δ be a high-level Golog program, and $\langle s_h, s_l\rangle \in B$. Then*

1. *if there is a situation s_l' in M_l s.t. $M_l, v[s/s_l, s'/s_l'] \models Do(m(\delta), s, s')$, then there is a situation s_h' in M_h s.t. $M_h, v[s/s_h, s'/s_h'] \models Do(\delta, s, s')$ and $\langle s_h', s_l'\rangle \in B$;*
2. *if there is an infinite history in M_l generated by $m(\delta)$ and satisfying C_l, then there is an infinite history in M_h generated by δ and satisfying C_h.*

4.4 Sound/Complete Abstractions on Theory Level

For both sound and complete abstractions, we first define their weak versions.

Definition 9. $\mathcal{G}_h$ is a weak sound m-abstraction of $\mathcal{G}_l$, if for any model M_l of $\mathcal{D}_l$, there is a model M_h of $\mathcal{D}_h$ s.t.

1. M_h is a sound m-abstraction of M_l via B;
2. for any situations s_h in M_h and s_l in M_l, if $\langle s_h, s_l\rangle \in B$ and $M_h, v[s_h/s] \models G_h[s]$, then $M_l, v[s_l/s] \models G_l[s]$.

In case that $G_l \equiv m(G_h)$, by Prop. 1, the above Condition 2 holds, and thus it can be removed from the definition.

Sound abstractions of theories require that each low-level model should have both sound and complete abstractions.

Definition 10. $\mathcal{G}_h$ is a sound m-abstraction of $\mathcal{G}_l$, if it is a weak sound m-abstraction of $\mathcal{G}_l$, and for any model M_l of $\mathcal{D}_l$, there is a model M_h of $\mathcal{D}_h$ s.t.

1. M_h is a complete m-abstraction of M_l via B;
2. for any situations s_h in M_h and s_l in M_l, if $\langle s_h, s_l\rangle \in B$ and $M_h, v[s_h/s] \models G_h[s]$, then $M_l, v[s_l/s] \models G_l[s]$.

Theorem 1. *Let $\mathcal{G}_h$ be a weak sound m-abstraction of $\mathcal{G}_l$. If δ is a weak solution to $\mathcal{G}_h$, so is $m(\delta)$ to $\mathcal{G}_l$.*

Proof. Let M_l be a model of $\mathcal{D}_l$. Then there is a model M_h of $\mathcal{D}_h$ s.t. M_h is a sound abstraction of M_l via B. We show that $M_l \models \text{Achieve}(m(\delta), G_l)$. Since $M_h \models \text{Achieve}(\delta, G_h)$, there is a situation s_h in M_h s.t. $M_h, v[s/s_h] \models Do(\delta, S_0, s) \wedge G_h[s]$. By Prop. 2, there is a situation s_l in M_l s.t. $M_l, v[s/s_l] \models Do(m(\delta), S_0, s)$ and $\langle s_h, s_l\rangle \in B$. By Condition 2 of Definition 9, $M_l, v[s/s_l] \models G_l[s]$. □

Theorem 2. *Let $\mathcal{G}_h$ be a sound m-abstraction of $\mathcal{G}_l$. If δ is a strong solution to $\mathcal{G}_h$, so is $m(\delta)$ to $\mathcal{G}_l$.*

Proof. By Theorem 1, $m(\delta)$ is weak solution to $\mathcal{G}_l$. We now prove it is also a strong solution. Let M_l be a model of $\mathcal{D}_l$. Then there is a model M_h of $\mathcal{D}_h$ s.t. M_h is a complete abstraction of M_l via B. We first show that $M_l \models \text{Ensure}(m(\delta), G_l)$. Let s_l be a situation in M_l s.t. $M_l, v[s/s_l] \models Do(\delta, S_0, s)$. By Prop. 3, there is a situation s_h in M_h s.t. $M_h, v[s/s_h] \models Do(\delta, S_0, s)$ and $\langle s_h, s_l\rangle \in B$. Since $M_h \models \text{Ensure}(\delta, G_h)$, $M_h, v[s/s_h] \models G_h[s]$. By Condition 2 of Definition 10, $M_l, v[s/s_l] \models G_l[s]$. We now show that $M_l \models \text{Term}(m(\delta), S_0, C_l)$. Assume that h_l is an infinite history in M_l generated by $m(\delta)$ and satisfying C_l. By Prop.

3, there is an infinite history in M_h generated by δ and satisfying C_h, contradicting with $M_h \models \text{Term}(\delta, S_0, C_h)$. □

Example 1 cont'd. It is easy to prove that $\mathcal{G}_h$ is a sound abstraction of $\mathcal{G}_l$. The following is a strong solution to $\mathcal{G}_h$:

$[(n(A) > 0 \wedge \textit{battery-level} \geq 10)?; \textit{clearablock}(A) \mid (\textit{battery-level} < 10)?;\ \textit{charge}]^*$.

Then, by Theorem 2, we get a strong solution of $\mathcal{G}_l$ as follows:

$[(\exists x.\textit{on}^+(x, A) \wedge \textit{battery-level} \geq 10)?;$
$\pi y, z.(\textit{on}(y, z) \wedge \textit{on}^*(z, A))?\ \textit{unstack}(y, z); \textit{drop}(y); \mid$
$(\textit{battery-level} < 10)?;\ \textit{charge}]^*$.

Definition 11. $\mathcal{G}_h$ is a weak complete m-abstraction of $\mathcal{G}_l$, if for any model M_h of $\mathcal{D}_h$, there is a model M_l of $\mathcal{D}_l$ s.t.

1. M_h is a complete m-abstraction of M_l via B;
2. for any situations s_h in M_h, s_l in M_l, if $\langle s_h, s_l\rangle \in B$ and $M_l, v[s_l/s] \models G_l[s]$, then $M_h, v[s_h/s] \models G_h[s]$.

Complete abstractions of theories require that every high-level model is both a sound abstraction of a low-level model and a complete abstraction of a low-level model.

Definition 12. $\mathcal{G}_h$ is a complete m-abstraction of $\mathcal{G}_l$, if it is a weak complete m-abstraction of $\mathcal{G}_l$, and for any model M_h of $\mathcal{D}_h$, there is a model M_l of $\mathcal{D}_l$ s.t.

1. M_h is a sound m-abstraction of M_l via B;
2. for any situations s_h in M_h, s_l in M_l, if $\langle s_h, s_l\rangle \in B$ and $M_l, v[s_l/s] \models G_l[s]$, then $M_h, v[s_h/s] \models G_h[s]$.

Theorem 3. *Let $\mathcal{G}_h$ be a weak complete m-abstraction of $\mathcal{G}_l$. If $m(\delta)$ is a weak solution to $\mathcal{G}_l$, so is δ to $\mathcal{G}_h$.*

Theorem 4. *Let $\mathcal{G}_h$ be a complete m-abstraction of $\mathcal{G}_l$. If $m(\delta)$ is a strong solution to $\mathcal{G}_l$, so is δ to $\mathcal{G}_h$.*

Corollary 1. *Let $\mathcal{G}_h$ be a sound and complete m-abstraction of $\mathcal{G}_l$. Then δ is a strong solution to $\mathcal{G}_h$ iff $m(\delta)$ is a strong solution to $\mathcal{G}_l$.*

5 Formalizing Existing Abstraction Works

In this section, we show that three main abstraction works in generalized planning can be formalized in our framework.

5.1 Srivastava et al.'s Work

QNP problems are classical planning problems extended with a set of non-negative numerical variables whose values are real numbers and can be decreased or increased randomly.

Definition 13. Let X be a set of non-negative numeric variables. $\mathcal{L}_X$ denotes the class of all consistent sets of literals of the form $x = 0$ and $x \neq 0$, for $x \in X$. A QNP problem $\mathcal{Q} = \langle X, I, G, O\rangle$ consists of X; $I \in \mathcal{L}_X$, a set of initially true literals; $G \in \mathcal{L}_X$, a set of goal literals; and O, a set of action operators. Every $o \in O$ has a set of preconditions $pre(o) \in \mathcal{L}_X$, and a set $eff(o)$ of effects of the form $inc(x)$ or $dec(x)$ for $x \in X$. An instance of $\mathcal{Q}$ is a quantitative planning problem whose initial state, which specifies a non-negative real number for each $x \in X$, satisfies I.

Definition 14. Let $\mathcal{Q} = \langle X, I, G, O\rangle$ be a QNP problem. A policy for $\mathcal{Q}$ is a mapping from qualitative states to actions. For $\epsilon > 0$, a trajectory of states is ϵ-bounded, if for any action o performed, if o decreases a variable x, then either the old value is $< \epsilon$ and the new value equals 0 or the amount of decrease is $\geq \epsilon$. A policy π solves an instance of $\mathcal{Q}$ if for any $\epsilon > 0$, every ϵ-bounded trajectory induced by π is goal reaching. A policy π solves $\mathcal{Q}$ if it solves every instance of it.

FOND problems are like classical planning problems except that an action o may have non-deterministic effects expressed as $eff_1(o)|\ldots|eff_n(o)$.

Definition 15. A QNP problem $\mathcal{Q} = \langle X, I, G, O\rangle$ is abstracted into a FOND problem $\mathcal{Q}'$ as follows: the literal $x \neq 0$ is replaced by an atom p_x; effect $inc(x)$ is replaced by deterministic effect p_x; and effect $dec(x)$ is replaced by non-deterministic effect $p_x|\neg p_x$.

Theorem 3 in [Srivastava *et al.*, 2011]. π is a policy that solves $\mathcal{Q}$ iff it is a strong cyclic policy for $\mathcal{Q}'$ that terminates.

Srivastava *et al.* proposed the Sieve algorithm to determine if a policy terminates. They made the key observation: In any ϵ-bounded trajectory, no variable can be decreased infinitely often without an intermediate increase. Based on this observation, Bonet and Geffner [2018] formalized the concept of conditional fairness. A strong cyclic policy for $\mathcal{Q}'$ that terminates can be equivalently defined as: a policy π s.t. every conditional fair trajectory induced by π is goal reaching.

Definition 16. A trajectory of states of $\mathcal{Q}'$ is conditionally fair if: from any time point on, for any $x \in X$, if no action with effect p_x ever occurs and actions with effect $p_x|\neg p_x$ occur infinitely often, then eventually p_x falsifies.

We now formalize the above work in our framework:

The low-level language is as follows: for each $x \in X$, we have a functional fluent $f_x(s)$; for each $o \in O$, we have an action A_o and a primitive action $a_o(\vec{\eta}_o)$, where $\vec{\eta}_o$ contains a parameter η_x for each $x \in X$ s.t. $inc(x)$ or $dec(x)$ is in $eff(o)$. The following is the specification of the g-planning problem $\mathcal{G}_q$. We omit the initial KB and the goal condition.

Precondition axioms: for each action $o \in O$,

$$Poss(a_o(\vec{\eta}_o), s) \equiv \bigwedge_{x=0 \in pre(o)} f_x(s) = 0 \wedge \bigwedge_{x\neq 0 \in pre(o)} f_x(s) \neq 0 \wedge \bigwedge_{\eta_x \in \vec{\eta}_o} \eta_x > 0$$

Successor state axioms: for each $x \in X$,

$$f_x(do(a, s)) = r \equiv \bigvee_{o\in O\ s.t.\ inc(x)\in eff(o)} \exists \vec{\eta}_o.\ a = a_o(\vec{\eta}_o) \wedge r = f_x(s) + \eta_x \vee \bigvee_{o\in O\ s.t.\ dec(x)\in eff(o)} \exists \vec{\eta}_o.\ a = a_o(\vec{\eta}_o) \wedge r = f_x(s) - \eta_x$$

Non-deterministic action definitions: for each $o \in O$,

$$A_o \equiv \pi \vec{\eta}_o.\ a_o(\vec{\eta}_o)$$

Trajectory constraint (ϵ-boundedness):

$$\exists \epsilon > 0 \forall s \sqsubset h \forall s' \bigwedge_{(x,o)\in S} \forall \vec{\eta}_o.s = do(a_o(\vec{\eta}_o), s') \supset f_x(s') < \epsilon \wedge f_x(s) = 0 \vee \eta_x \geq \epsilon,$$

where $S = \{(x, o) \mid x \in X, o \in O, dec(x) \in eff(o)\}$.

The high-level language is as follows: for each $x \in X$, we have a relational fluent $p_x(s)$, meaning $x > 0$; for each $o \in O$, we have an action B_o and a primitive action $b_o(\vec{\eta}_o)$, where each η_x takes values from $\{0, 1\}$, where 0 means p_x becomes false and 1 means true.

We omit the specification of the high-level g-planning problem $\mathcal{G}_f$ except the trajectory constraint for conditional fairness. We let $dec(x, s)$ abbreviate for the following formula, which means s results from an action which decreases x ($inc(x, s)$ can be similarly defined.):

$$\exists a \exists s'.s = do(a, s') \wedge \bigvee_{o\in O \, s.t. \, dec(x)\in eff(o)} \exists \vec{\eta}_o.a = b_o(\vec{\eta}_o).$$

Then conditional fairness is formalized as follows:
$\forall s \sqsubset h.\psi \supset \exists s' \sqsubset h.s \sqsubseteq s' \wedge \neg p_x(s')$, where ψ is

$$\forall s' \sqsubset h[s \sqsubseteq s' \supset \neg inc(x, s') \wedge \exists s''(s' \sqsubseteq s'' \wedge dec(x, s''))].$$

Below is the refinement mapping: for each $x \in X$, $m(p_x) = f_x > 0$; for each $o \in O$, $m(B_o) = A_o$.

Theorem 5. *$\mathcal{G}_f$ is a sound & complete m-abstraction of $\mathcal{G}_q$.*

Proof. Note that we have $G_q = m(G_f)$. To prove that a conditional fair high-level infinite history corresponds to a ϵ-bounded low-level infinite history, we use the technique in the proof of Theorem 5 (Completeness of the Sieve algorithm) in [Srivastava *et al.*, 2011]. □

5.2 Bonet and Geffner's Work

Bonet and Geffner [2018] proposed solving generalized classical planning problems by abstracting them into QNP problems. For example, for the problem of achieving $clear(A)$, it can be abstracted into a QNP problem with one numeric feature $depth(A)$, meaning the number of blocks above A, an action that decrements $depth(A)$, and the goal $depth(A) = 0$. They showed that if the abstraction is sound, then a solution to the QNP problem is also a solution to the original problem. However, their abstract actions are restricted in the sense that the execution of an abstract action results in the execution of a single concrete action.

Using our framework, we can give a more general formalization of their work where the execution of an abstract action may correspond to that of a sequence of concrete actions. The language of a generalized classical planning problem $\mathcal{G}_c$ consists of a set of relational fluents and a set of deterministic actions. The trajectory constraint of $\mathcal{G}_c$ is simply $\top$. Such a problem is abstracted into a generalized numerical planning problem $\mathcal{G}_{bq}$ extended with a set of relational fluents $p(s)$. The refinement mapping maps a relational fluent to a low-level formula, a functional fluent to a low-level nat term, possibly a counting term, and maps an action A to a low-level program δ executable when the precondition of A holds and achieving the effect of A. $\mathcal{G}_{bq}$ is a sound abstraction of $\mathcal{G}_c$, if for every instance M_c of $\mathcal{G}_c$, which is a classical planning problem, there is an instance of $\mathcal{G}_{bq}$, which is a quantitative numerical planning problem, s.t. M_{bq} is a sound abstraction of M_c. Then we have the following result:

Corollary 2. *If $\mathcal{G}_{bq}$ is a sound abstraction of $\mathcal{G}_c$ and δ is a strong solution to $\mathcal{G}_{bq}$, then $m(\delta)$ is a strong solution to $\mathcal{G}_c$.*

5.3 Illanes and McIlraith's Work

Illanes and McIlraith[2019] considered solving a class of generalized classical planning problems by automatically deriving a sound QNP abstraction from an instance of the problem. The automatic abstraction process is based on introducing a counter for each set of indistinguishable objects using the idea from [Fuentetaja and de la Rosa, 2016]. For example, suppose we have an instance where a number of packages must be delivered from a source location to either of two other locations A or B. Then we can automatically extract two counters, one for packages to be relocated to A, and one for packages to be relocated to B. Then the QNP problem is solved by converting it into a FOND problem, which they call a quantified planning problem. What distinguishes this work from that of [Bonet and Geffner, 2018] is that the latter didn't address the issue of automatical derivation of sound QNP abstractions.

Similarly to [Bonet and Geffner, 2018], Illanes and McIlraith's abstraction from generalized classical planning to QNP can be formalized in our framework as $\mathcal{G}'_c$ and $\mathcal{G}'_{bq}$, and the refinement mapping maps each functional fluent to a counting term. Then we have the following result:

Corollary 3. *$\mathcal{G}'_{bq}$ is a sound abstraction of $\mathcal{G}'_c$. Thus if δ is a strong solution to $\mathcal{G}'_{bq}$, then $m(\delta)$ is a strong solution to $\mathcal{G}'_c$.*

6 Conclusions

In this paper, we proposed a uniform abstraction framework for generalized planning based on the situation calculus and Golog. We formalized the concept of generalized planning problems and solutions, covering generalized classical planning, QNP, and FOND. We defined the concepts of sound/complete abstractions of generalized planning problems and show that solutions to a generalized planning problem are nicely related to those of its sound/complete abstractions. In this paper, we only give model-theoretic definitions of sound/complete abstractions. In the future, we are interested in their proof-theoretic characterizations. Further, we would like to explore automatic discovery of sound/complete abstractions.

Acknowledgments

We thank Yves Lespérance for helpful discussions on the paper. We acknowledge support from the Natural Science Foundation of China under Grant No. 62076261.

References

[Bacchus *et al.*, 1995] Fahiem Bacchus, Joseph Y. Halpern, and Hector J. Levesque. Reasoning about noisy sensors in the situation calculus. In *Proceedings of the Fourteenth*

International Joint Conference on Artificial Intelligence (IJCAI-95), pages 1933–1940, 1995.

[Banihashemi *et al.*, 2017] Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. Abstraction in situation calculus action theories. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 1048–1055, 2017.

[Bonet and Geffner, 2018] Blai Bonet and Hector Geffner. Features, projections, and representation change for generalized planning. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 4667–4673, 2018.

[Bonet *et al.*, 2017] Blai Bonet, Giuseppe De Giacomo, Hector Geffner, and Sasha Rubin. Generalized planning: Nondeterministic abstractions and trajectory constraints. In *Proceedings of the Twenty-sixth International Joint Conference on Artificial Intelligence*, 2017.

[De Giacomo *et al.*, 2000] Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. Congolog, a concurrent programming language based on the situation calculus. *Artif. Intell.*, 121(1-2):109–169, 2000.

[Fuentetaja and de la Rosa, 2016] Raquel Fuentetaja and Tomás de la Rosa. Compiling irrelevant objects to counters. Special case of creation planning. *AI Commun.*, 29(3):435–467, 2016.

[Hu and De Giacomo, 2011] Yuxiao Hu and Giuseppe De Giacomo. Generalized planning: Synthesizing plans that work for multiple environments. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 2011.

[Illanes and McIlraith, 2019] León Illanes and Sheila A. McIlraith. Generalized planning via abstraction: Arbitrary numbers of objects. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, pages 7610–7618, 2019.

[Immerman and Vardi, 1997] Neil Immerman and Moshe Y. Vardi. Model checking and transitive-closure logic. In *Proceeding of 9th International Conference on Computer Aided Verification*, pages 291–302, 1997.

[Kuske and Schweikardt, 2017] Dietrich Kuske and Nicole Schweikardt. First-order logic with counting. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–12, 2017.

[Levesque *et al.*, 1997] Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1–3):59–83, 1997.

[Li and Liu, 2020] Jian Li and Yongmei Liu. Automatic verification of liveness properties in the situation calculus. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, pages 2886–2892, 2020.

[Reiter, 2001] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

[Schulte and Delgrande, 2004] Oliver Schulte and James P. Delgrande. Representing von Neumann-Morgenstern games in the situation calculus. *Annals of Mathematics and Artificial Intelligence*, 42(1-3):73–101, 2004.

[Segovia-Aguas *et al.*, 2016] Javier Segovia-Aguas, Sergio Jiménez, and Anders Jonsson. Generalized planning with procedural domain control knowledge. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS)*, 2016.

[Srivastava *et al.*, 2008] Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. Learning generalized plans using abstract counting. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, 2008.

[Srivastava *et al.*, 2011] Siddharth Srivastava, Shlomo Zilberstein, Neil Immerman, and Hector Geffner. Qualitative numeric planning. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, pages 1010–1016, 2011.