Automatic Verification of Sound Abstractions for Generalized Planning

Zhenhe Cui, Weidu Kuang, Yongmei Liu
Dept. of Computer Science, Sun Yat-sen University, Guangzhou 510006, China
{cuizhh3, kuangwd}@mail2.sysu.edu.cn, ymliu@mail.sysu.edu.cn

Abstract
Generalized planning studies the computation of general solutions for a set of planning problems. Computing general solutions with correctness guarantee has long been a key issue in generalized planning. Abstractions are widely used to solve generalized planning problems. Solutions of sound abstractions are those with correctness guarantees for generalized planning problems. Recently, Cui et al. proposed a uniform abstraction framework for generalized planning. They gave the model-theoretic definitions of sound and complete abstractions for generalized planning problems. In this paper, based on Cui et al.’s work, we explore automatic verification of sound abstractions for generalized planning. We firstly present the proof-theoretic characterization for sound abstraction. Then, based on the characterization, we give a sufficient condition for sound abstractions which is first-order verifiable. To implement it, we exploit regression extensions, and develop methods to handle counting and transitive closure. Finally, we implement a sound abstraction verification system and report experimental results on several domains.

1 Introduction
Generalized planning, where a single plan works for a set of similar planning problems, has received continual attention in the AI community [Srivastava et al., 2008; Hu and De Giacomo, 2011; Bonet and Geffner, 2018; Segovia et al., 2019; Illanes and McIlraith, 2019]. For example, the general solution “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. Computing general solutions with correctness guarantee has long been a key issue in generalized planning. However, so far only limited correctness results have been obtained, mainly for the so-called one-dimensional problems [Hu and Levesque, 2010] and extended LL-domains [Srivastava et al., 2011a].

Abstractions are widely used to solve generalized planning problems. The idea is to develop an abstract model of the problem, which is easier to solve, and exploit a solution in the abstract model to find a solution in the concrete model. A popular kind of abstract models for generalized planning are qualitative numerical planning (QNP) problems, introduced by [Srivastava et al., 2011b]: they showed that QNPs are decidable, and proposed a generate-and-test method to solve QNPs. Bonet and Geffner [2018] proposed solving a generalized classical planning problem by abstracting it into a QNP, they showed that if the abstraction is sound, then a solution to the QNP is also a solution to the original problem. Here, soundness of abstractions is the key to guarantee correctness of solutions to the original problem.

The automatic generation of sound abstractions for generalized planning problems has attracted the attention of researchers in recent years. Bonet et al. [2019a] proposed learning a QNP abstraction for a generalized planning problem from a sample set of instances, however, the abstraction is guaranteed to be sound only for the sample instances. Further, Bonet et al. [2019b] showed how to obtain first-order formulas that define a set of instances on which the abstraction is guaranteed to be sound. Illanes and McIlraith [2019] considered solving a class of generalized 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, however, this class of problems is too restricted.

Recently, Banihashemi et al. [2017] proposed an agent abstraction framework based on the situation calculus [Reiter, 2001] and Golog [Levesque et al., 1997]. They related 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. Based on their work, Cui et al. [2021] proposed a uniform abstraction framework for generalized planning. They formalized a generalized planning problem as a triple of a basic action theory, a trajectory constraint, and a goal. They gave model-theoretic definitions of sound/complete abstractions of a generalized planning problem, and showed that solutions to a generalized planning problem are nicely related to those of its sound/complete abstractions. In particular, the refinement of any solution to a sound abstraction is a solution to the original problem.

The significance of the research line initiated by Bonet and Geffner [2018] is that abstract problems are easier to solve, soundness of abstractions together with correctness of high-
level solutions guarantee correctness of low-level solutions. In this paper, based on Cui et al.’s work, we explore automatic verification of sound abstractions for generalized planning. First of all, we give a proof-theoretic characterization of sound abstractions for generalized planning in the situation calculus. Based on the characterization, we give a sufficient condition for sound abstractions which is first-order verifiable. To verify the condition with first-order theorem provers, we exploit universal and existential extensions of regression, and develop methods to handle counting and transitive closure. Using the SMT solver Z3, we implemented a verification system and experimented with 7 domains: 5 from the literature and 2 made by us. Experimental results showed that our system was able to successfully verify soundness of abstractions for all domains in seconds.

2 Preliminaries
2.1 Situation Calculus 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 $s \subseteq s'$, 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 $\phi$, we use $\phi[s]$ to denote the formula obtained from $\phi$ by restoring $s$ as the situation arguments to all fluents. A situation $s$ is executable if it is possible to perform the actions in $s$ one by one: $Exec(s) \equiv \forall s.a,s'.do(a,s') \subseteq s \supset Poss(a,s')$.

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

$$D = \Sigma \cup D_{ap} \cup D_{ss} \cup D_{una} \cup D_{S_0},$$

where $\Sigma$ is the set of the foundational axioms for situations; $D_{ap}$ 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 $s$; $D_{ss}$ is a set of successor state axioms (SSAs), 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 $f(\vec{x},do(a,s)) = y \equiv \psi_f(\vec{x},y,a,s)$, where $\phi_P(\vec{x},a,s)$ and $\psi_f(\vec{x},y,a,s)$ are uniform in $s$; $D_{una}$ is the set of unique name axioms for actions; $D_{S_0}$ is the initial knowledge base stating facts about $S_0$.

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

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

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

Golog has two kinds of semantics [De Giacco et al., 2000]: transition semantics and evaluation semantics. In transition semantics, a configuration of a Golog program is a pair $(\delta,s)$ of a situation $s$ and a program $\delta$ remaining to be executed. The predicate $Trans(\delta,s,d,s')$ means that there is a transition from configuration $(\delta,s)$ to $(d',s')$ in one elementary step. The predicate $Final(\delta,s)$ means that the configuration $(\delta,s)$ is a final one, which holds if program $\delta$ may legally terminate in situation $s$. In evaluation semantics, the predicate $Do(\delta,s,s')$ means that executing the program $\delta$ in situation $s$ will terminate in a situation $s'$. Do can be defined with $Trans$ and $Final$ as follows: $Do(\delta,s,s') \equiv \exists d'.Trans'((\delta,s,d,s') \land Final(d',s'))$, where, $Trans'$ denotes the transitive closure of $Trans$.

2.2 Regression and Its Extensions

Regression is an important computational mechanism for reasoning about deterministic actions and their effects in the situation calculus [Reiter, 2001]. The following is the definition of a one-step regression operator:

**Definition 1.** Given a BAT $D$ and a formula $\phi$. We use $R_D[\phi]$ to denote the formula obtained from $\phi$ by the following steps: for each term $f(\vec{t},do(a,\sigma))$, replace $\phi$ with $\exists y.\psi_f(\vec{t},y,\alpha,\sigma) \land \phi[f(\vec{t},do(a,\sigma))/y]^1$; replace each atom $P(\vec{t},do(\alpha,\sigma))$ with $\phi_P(\vec{t},\alpha,\sigma)$; replace each precondition atom $Poss(A(\vec{t},\sigma)) \Pi_A(\vec{t},\sigma)$; and further simplify the result with $D_{una}$.

**Proposition 1.** $D \models \phi \equiv R_D[\phi]$.

Luo et al. [2020] presented the existentially extended regression, notation $R_E[\phi(s),\delta]$ denotes a state formula expressing that there exists an execution of program $\delta$ starting from $s$ making $\phi$ hold.

**Definition 2.** Given a situation-suppressed formula $\phi$ and a Golog program $\delta$, the extended regression $R_E[\phi(s),\delta]$ can be inductively defined as follows:

- $R_E[\phi(s),\alpha] = R_D[\Pi_A(\vec{x},s) \land \phi(do(\alpha,\sigma))]$,
- $R_E[\phi(s),\psi?'] = \psi[s] \land \phi(s)$,
- $R_E[\phi(s),\delta_1 ; \delta_2] = R_E[\phi(s),\delta_2] \land R_E[\phi(s),\delta_1]$,
- $R_E[\phi(s),\pi x.\delta(x)] = (\exists x)R_E[\phi(s),\delta(x)]$.

**Proposition 2.** Given a basic action theory $D$, a Golog program $\delta$ and a situation-suppressed formula $\phi$, we have:

$$D \models R_E[\phi(s),\delta] \equiv \exists s'.Do(\delta,s,s') \land \phi[s'].$$

Li and Liu [2015] presented the universally extended regression, notation $R_U[\phi(s),\delta]$ denotes a state formula expressing that all executions of program $\delta$ starting from $s$ making $\phi$ hold. To get the definition of universally extended regression, one can replace the symbols $R_E$, $\land$, $\lor$, and $\exists$ in Definition 2 with $R_U$, $\lor$, $\land$, and $\forall$, respectively.

$^1\phi[t'/t]$ denotes that formula obtained from $\phi$ by replacing all occurrences of $t'$ in $\phi$ by $t$. 
Proposition 3. Given a basic action theory $D$, a Golog program $\delta$ and a situation-suppressed formula $\phi$, we have:

$$D \models R^U[\phi(s), \delta] \equiv \forall s'.[Do(\delta, s, s') \supset \phi(s')]$$

2.3 Generalized Planning Abstraction Framework

Situation calculus cannot express the property of termination, counting, transitive closure, and non-deterministic actions. Cui et al. [2021] extended the situation calculus for these four aspects: to represent the property of termination, following [Schulte and Delgrande, 2004], they use the situation calculus with infinite histories; to represent planning with non-deterministic actions, they treat a non-deterministic action as a non-deterministic program; to extend the situation calculus with counting, following the logic FOCN [Kuske and Schweikardt, 2017], they introduce counting terms of the form $\#y.\varphi(x)$ meaning that the block $\varphi$ transitive closure is often used to define counting terms, following transitive closure logic [Immerman and Vardi, 1997], they introduced the notation $[TC_{x,y}\varphi(u,v)]$, where $\varphi(x, y)$ is a formula with 2k-free variables, $u$ and $v$ are two k-tuples of terms, which says that the pair $(u, v)$ is contained in the reflexive transitive closure of the binary relation on k-tuples that is defined by $\varphi$.

In the following, we introduce the generalized planning abstraction framework proposed in [Cui et al., 2021].

Definition 3. A generalized planning problem is a triple $G = \langle D, C, G \rangle$, where $D$ is a BAT, $C$ is a trajectory constraint, i.e., a situation calculus formula with a free variable of infinite histories, and $G$ is a goal condition.

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.

Example 1. In the blocks world, an agent can perform two kinds of actions: $mt(x)$ (move x to the table), provided x is being held, and $unstack(x,y)$ (unstack x from y, provided x is clear and x is on y). There are four fluent: $\text{clear}(x)$, $\text{ontable}(x)$, $\text{on}(x,y)$, and $\text{holding}(x)$. In this problem, the trajectory constraint $C$ is $\top$, the goal state $G$ is $\text{clear}(A)$, some example axioms from $D$ are as follows:

Precondition Axioms:

$$\text{Poss}(mt(x),s) \equiv \text{clear}(x,s) \land \neg\text{ontable}(x,s).$$

Successor State Axioms:

$$\text{on}(x,y,do(a,s)) \equiv \text{on}(x,y,s) \land \neg a = \text{unstack}(x,y).$$

Initial Situation Axiom:

$$\exists x.\text{ont}(x,A) \land \text{ontable}(A) \land \neg\text{holding}(x),$$

where the formula $\text{ont}(x,A)$ is a transitive closure formula with a concise form, which means that the block x is above the block A.

Abstractions for generalized planning problems are specified by the following notion of refinement mapping:

Definition 4. A function $m$ is a refinement mapping from $Gh = \langle D_h, C_h, Gh \rangle$ to $Gl = \langle D_l, Cl, Gl \rangle$ if for each HL action type $A$, $m(A(x)) = \delta_A(x)$, where $\delta_A(x)$ is a LL program; for each HL relational fluent $P$, $m(P(x)) = \phi_P(x)$, where $\phi_P(x)$ is a LL situation-suppressed formula; for each HL functional fluent $f$, $m(f(x)) = \tau_f(x)$, where $\tau_f(x)$ is a LL (counting) term.

Given a refinement mapping $m$, they introduced an isomorphism relation, called $m$-isomorphic, between a HL and a LL situation as follows:

Definition 5. Given a refinement mapping $m$, a situation $s_h$ of a HL model $M_h$ is $m$-isomorphic to a situation $s_l$ in a LL model $M_l$, written $s_h \sim_m s_l$, if: for any HL relational fluent $P$, and variable assignment $v$, we have $M_h[v/s_h] \models P(x, s) \iff M_l[v/s_l] \models m(P)(x, s)$; for any HL functional fluent $f$, variable assignment $v$, we have $M_h[v/s_h] \models f(x, s) = y \iff M_l[v/s_l] \models m(f)(x, s) = y$.

To relate HL and LL models, they defined two relations: $m$-simulation and $m$-back-simulation. Intuitively, $m$-simulation means: whenever a refinement of a HL action can occur, so can the HL action, and back-simulation means the other direction. Here we only present the definition of $m$-simulation relation, $m$-back-simulation relation can be defined symmetrically. In the following, $\Delta^M_S$ denotes the situation domain of $M$; $S_0^M$ stands for the initial situation of $M$; the notation $\text{Term}(\delta, s, C)$ means starting in situation $s$, program $\delta$ terminates under constraint $C$:

$$\text{Term}(\delta, s, C) \equiv \neg\exists h(C(h) \land s'/h \exists\delta')\text{Trans}^+(\delta, s, \delta', s').$$

Definition 6. A relation $B \subseteq \Delta^M_S \times \Delta^M_S$ is an $m$-simulation relation between $M_h$ and $M_l$, if $\langle s_0^M, s_0^M \rangle \in B$ and the following hold: (1) $\langle s_h, s_l \rangle \in B$ implies that: $s_h \sim_m s_l$; for any HL action type $A$, and variable assignment $v$, $M_h[v/s_h] \models \text{Term}(m(A(x)), s, C_l)$, and if there is a situation $s_l'$ s.t. $M_l[v/s_l, s'/s_l'] \models \text{Do}(m(A(x)), s, s')$, then there is a situation $s_h'$ s.t. $M_h[v/s_h, s'/s_h'] \models \text{Do}(A(x), s, s')$ and $\langle s_h', s_l \rangle \in B$. (2) For any infinite HL action sequence $\sigma$, 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 $\sigma$ and satisfying $C_h$.

Based on the notions above, they defined sound/completeness abstraction on model and theory level. For these two levels, sound abstractions mean that HL behavior entails LL behavior, complete abstractions mean the other direction.

Definition 7. $M_h$ is a sound $m$-abstraction of $M_l$, if there is an $m$-back-simulation relation $B$ between $M_h$ and $M_l$.

Definition 8. $M_h$ is a complete $m$-abstraction of $M_l$, if there is a $m$-simulation relation $B$ between $M_h$ and $M_l$.

On the theory level, for sound abstraction, they defined two versions (complete abstractions can be defined symmetrically, and we omit here):

Definition 9. $G_h$ is a weak sound $m$-abstraction of $G_l$, if for any model $M_l$ of $D_l$, there is a model $M_h$ of $D_h$ such that: (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 $M_h[v/s_h] \models G_h[s]$, then $M_l[v/s_l] \models G_l[s]$.

Definition 10. $G_h$ is a sound $m$-abstraction of $G_l$, if it is a weak sound $m$-abstraction of $G_l$, and for any model $M_l$ of $D_l$, there is a model $M_h$ of $D_h$ such that: (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 $M_h[v/s_h] \models G_h[s]$, then $M_l[v/s_l] \models G_l[s]$. 
3 Proof-Theoretic Characterization

In this section, we give a proof-theoretic characterization for sound abstractions for generalized planning (g-planning).

First of all, we introduce some notations and conventions. We define the program of doing any HL action sequence and its refinement as follows:

\[ \text{anyhlas} \equiv \langle \alpha \in A_h \wedge \exists \vec{x}. A(\vec{x}) \rangle^*, \text{angllps} \equiv m(\text{anyhlas}). \]

We call a situation \( s \) a s.t. Do(anyllps, \( S_0, s \)) holds an executable refinement of a HL situation.

The notation \( \text{Infs}\phi(\delta, h, C) \) means \( h \) is an infinite execution of program \( \delta \) satisfying trajectory constraint \( C \):

\[ \text{Infs}\phi(\delta, h, C) = C(h) \wedge \forall s' \exists h \exists h'. \text{Trans}\phi^*(\delta, h', \delta'). \]

We introduce an abbreviation \( R(s, s') \), which means that situations \( s \) and \( s' \) result from executing the refinement of the same HL action sequence:

\[ R(s, s') \equiv \forall P. P(S_0, S_0) \wedge \\\bigwedge_{\alpha \in A_h} \forall \vec{x}. s_1, s_1', (\exists (s, s') \wedge \text{Do}(m(A(\vec{x}))), s_1, s_1') \wedge \text{Do}(m(A(\vec{x}))), s_1', s_1'') \supset P(s_1', s_1'') \supset P(s, s'). \]

Let \( \phi \) be a HL formula uniform in a situation. We use \( m(\phi) \) to denote the formula resulting from replacing each high-level symbol in \( \phi \) with its LL definitions. We now define \( m(C) \) for a HL trajectory constraint \( C \). For this purpose, we first define a normal form for trajectory constraints.

**Definition 11.** Let \( C \) be a trajectory constraint. We say that \( C \) is in normal form if \( C \) contains no occurrence of action variables or Poss, and any appearance of do must be in the form of \( s' = \text{do}(\vec{A}(\vec{t}), s) \), where \( s \) and \( s' \) are variables.

It is easy to prove that any trajectory constraint can be converted to an equivalent one in normal form.

**Definition 12.** Let \( C \) be a HL trajectory constraint. We first convert it into an equivalent one \( C' \) in normal form. We let \( m(C) \) denote the LL constraint obtained from \( C' \) as follows: first replace any appearance of \( \exists s . \) Do(angllps, \( S_0, s \)) then replace any appearance of \( s' = \text{do}(\vec{A}(\vec{t}), s) \) with \( \text{Do}(m(\vec{A}(\vec{t})), s, s') \), and finally replace any high-level symbols with its LL definitions.

We now extend the \( m \)-simulation or \( m \)-back-simulation relation \( B \) to infinite histories, and show that if a HL infinite history \( h_h \) and a LL infinite history \( h_l \) are \( B \)-related, then \( h_h \) satisfies a constraint \( C \) iff \( h_h \) satisfies \( m(C) \).

**Definition 13.** Let \( M_h \) be a sound or complete abstraction of \( M_l \) via \( B \). Let \( h_h \) and \( h_l \) be finite histories of \( M_h \) and \( M_l \), respectively. We write \( \langle h_h, h_l \rangle \in B \), if for any \( s_h \cap h_h \), there exist \( s_l \subset h_l \), such that \( \langle s_h, s_l \rangle \in B \).

By induction on the structure of a normal form trajectory constraint, we can prove:

**Proposition 4.** Let \( M_h \) be a sound and complete abstraction of \( M_l \). Let \( C \) be a HL trajectory constraint. Then \( M_h \models C \) iff \( M_l \models m(C) \).

**Proof.** We do the proof by using structural induction on \( C \):

**Induction base:**

- \( C \vdash s = s' \): Let \( s_h \) and \( s''_h \) are two situations in \( M_h \), and \( C \) be the atom \( s = s'_h \). On one hand, if we have \( M_h, v[s/s_h, s''/s_h'] \models s = s' \), then there exists a HL grounded action sequence \( \alpha \), such that \( M_h, v[s/s_h, s''/s_h'] \models s = \text{do}(\alpha, S_0) \wedge s'' = \text{do}(\alpha, S_0) \). Based on the definition of sound abstraction on model level, we have that there exist two situations \( s_l \) and \( s'_l \) such that \( M_h, v[s/s_h, s''/s_h''] \models \text{do}(\alpha, S_0) \wedge \text{do}(m(\alpha), S_0, s) \). Given the Definition 12, we can get \( M_l \models m(s_h = s'_h) \).

- \( C \vdash s \subset s' \): Let \( s_h \) and \( s''_h \) are two situations in \( M_h \), and \( C \) be the atom \( s \subset s'_h \). On one hand, if we have \( M_l \models m(s_h \subset s''_h) \), then for the HL action sequence \( \alpha \), based on the Definition 12, we have \( M_l, v[s/s_l, s''/s_l'] \models \text{do}(\alpha, S_0) \wedge \text{do}(m(\alpha), S_0, s) \wedge \text{do}(m(\alpha), S_0, s) \). By using the definition of complete abstraction on model level, we have that there exist two situations \( s_l \) and \( s''_l \) such that \( M_h, v[s/s_h, s''/s_h''] \models s = \text{do}(\alpha, S_0) \wedge s'' = \text{do}(\alpha, S_0) \). Then we can get \( M_l \models m(s_h \subset s''_h) \).

- \( C \vdash s \cap h \): Let \( s_h \) be a situation in \( M_h \), \( h_h \) be an infinite history in \( M_h \), and \( C \) be the atom \( s \cap h_h \). On one hand, there exists an infinite HL grounded action sequence \( \sigma \) and a finite subsequence \( \alpha \) of \( \sigma \), such that \( M_h, v[s/s_h, h/h_h] \models s = \text{do}(\alpha, S_0) \wedge s \subset h_h \). Using the action sequence \( \sigma \), and based on the definition of sound abstraction on model level, we can easily construct an infinite history \( h_l \) in \( M_l \) corresponds to \( h_h \), and there also exists a situation \( s_l \subset h_l \), such that \( M_l, v[s/s_l, h/h_l] \models \text{do}(m(\alpha), S_0, s) \wedge s \subset h_l \). Given the Definition 12, we can get \( M_l \models m(s_h \cap s_l) \).

On the other hand, if we have \( M_l \models m(s_h \cap s_l) \), then for the HL action sequences \( \sigma \) and \( \alpha \), based on the Definition 12, we have \( M_l, v[s/s_l, h/h_l] \models \text{do}(m(\alpha), S_0, s) \wedge s \subset h_l \). By using the definition of complete abstraction on model level, we have that there exist a situation \( s_h \) and an infinite history \( h_h \) generated by \( \sigma \) such that \( M_h, v[s/s_h, h/h_h] \models s = \text{do}(\alpha, S_0) \wedge s \subset h_h \). Then we can get \( M_l \models m(s_h \cap s_l) \).
• $C \models a = a'$: Let $a$ and $a'$ be two HL (universal) primitive deterministic actions, and $C$ be the atom $a = a'$. Then based on the definition of refinement mapping, we have that $M_h \models a = a' \iff M_l \models m(a) = m(a') \iff M_l \models m(a) = m(a')$.

• $C \models P(\bar{x}, s)$: Let $s_h$ be a situation in $M_h$, $P$ is a HL relational fluent, and $C$ be the atom $P(\bar{x}, s)$. Then based on the definition of $m$-isomorphic, we have that there exists a situation $s_l$ in $M_l$ such that $M_h, v[s/s_h] \models P(\bar{x}, s) \iff M_l, v[s/s_l] \models m(P)(\bar{x}, s)$.

• $C \models f(\bar{x}, s) = y$: Let $s_h$ be a situation in $M_h$, $f$ is a HL relational fluent, and $C$ be the atom $f(\bar{x}, s) = y$. Then based on the definitions of simulation and $m$-isomorphic, we have that there exists a situation $s_l$ in $M_l$ such that $M_h, v[s/s_h] \models f(\bar{x}, s) = y \iff M_l, v[s/s_l] \models m(f)(\bar{x}, s) = y$.

**Induction step:** Let $C_1$ and $C_2$ be two HL trajectory constraints, and suppose that

$$M_h \models C_1 \text{ iff } M_l \models m(C_1), \quad M_h \models C_2 \text{ iff } M_l \models m(C_2).$$

Then we prove the following three cases:

• $C \models \neg C_1$: If we have $M_h \models \neg C_1$, then we can get $M_l \not\models m(C_1)$, which means $M_l \models \neg m(C_1)$. Thus, we have $M_l \models \neg m(C_1)$.

• $C \models C_1 \land C_2$: If we have $M_h \models C_1 \land C_2$, then we can get $M_l \models m(C_1)$ and $M_l \models m(C_2)$, which means $M_l \models m(C_1) \land m(C_2)$. Thus, we have $M_l \models m(C_1) \land m(C_2)$.

• $C \models \exists s.C_1(s)$: If we have $M_h \models \exists s.C_1(s)$, then there exists a situation $s_h$ in $M_h$ reached from the initial situation $s_h$ via executing an action sequence $\alpha$, such that $M_h, v[s/s_h] \models C_1(s)$. Based on the definition of sound abstraction on model level and inductive assumption, we have that there exists a situation $s_l$ in $M_l$ such that $M_l, v[s/s_l] \models Do(m(\alpha), s_h, s) \land m(C_1)(s)$, which means $M_l \models m(\exists s.C_1(s))$.

Non-deterministic actions in [Cui et al., 2021] are treated as non-deterministic programs. In particular, each non-deterministic action $A$ has a definition in the form $A(\bar{x}) = \pi \bar{u}.A_d(\bar{x}, \bar{u})$, where $A_d$ is a deterministic action. We let $\Pi_\alpha(\bar{x}, s)$ denote $\exists \bar{u}.\Pi_{A_d}(\bar{x}, \bar{u}, s)$, let $\phi_{P, A_d}(\bar{y}, \bar{x}, \bar{u}, s)$ denote $\phi_P(\bar{y}, \bar{x}, \bar{u}, s)$ simplified by using $\Delta_{una}$, and let $\psi_{f, A_d}(\bar{y}, \bar{z}, \bar{x}, \bar{u}, s)$ denote $\psi_P(\bar{y}, \bar{z}, m(\bar{A}_d(\bar{x}, \bar{u}, s)))$ simplified by using $\Delta_{una}$.

We now introduce the following abbreviations:

$$\psi_T \triangleq \bigwedge_{A \in A_h} \forall \bar{x}.\text{Term}(m(A(\bar{x})), s, C_1),$$

$$\xi_P \triangleq \bigwedge_{A \in A_h} \forall \bar{x}, s'. Do(m(A(\bar{x})), s, s') \supset \exists \bar{u}.\bigwedge_{P \in P_h} [\forall \bar{y}. m(P(\bar{y}, s')) \equiv m(\phi_{P, A_d}(\bar{y}, \bar{x}, \bar{u}, s))],$$

$$\xi_f \triangleq \bigwedge_{A \in A_h} \forall \bar{x}, s'. Do(m(A(\bar{x})), s, s') \supset \exists \bar{u}.\bigwedge_{f \in F_h} [\forall \bar{y}, z. m(f(\bar{y}, s')) = z \equiv m(\psi_{f, A_d}(\bar{y}, z, \bar{x}, \bar{u}, s))],$$

where $\psi_T$ says that the refinement of any HL action termi-

nates in $s$ under $C_1$; $\xi_P$ and $\xi_f$ says that for any HL action $A(\bar{x})$, if its refinement transforms situation $s$ to $s'$, then there is $\bar{u}$ s.t. the mapping of all SSAs instantiated with $A_d(\bar{x}, \bar{u})$ hold for $s$ and $s'$.

The following theorem gives a proof-theoretic characterization for sound abstractions, where Item 1 and Item 6 are easy to understand; Item 2 says that $D_1$ entails that for any executable refinement of a HL situation, the executability of the refinement of any HL action implies its mapped precondition; Item 3 says that $D_1$ entails that for any executable refinement of a HL situation, the mapped precondition of any HL action implies that the executability of its refinement holds in some $R$-related situation; and Item 5 says that $D_1$ entails that the existence of an infinite execution of any pistols satisfying the LL constraint is equivalent to the existence of one satisfying the mapped HL constraint.

**Theorem 1. (Sound abstraction)** Given a generalized planning problem $G_1$ and its abstraction $G_h$, $G_h$ is a sound m-
abstraction of $G_1$ iff the following hold:

1. $D_1 \models \psi_T$, where $\phi \in D_h^\phi$.
2. $D_1 \models \forall s. Do(\text{anyllps}, s_0, s) \supset \bigwedge_{A \in A_h} \forall \bar{x}, s'. Do(m(A(\bar{x})), s, s') \supset m(\Pi(\bar{x}, s));$
3. $D_1 \models \forall s. Do(\text{anyllps}, s_0, s) \supset \bigwedge_{A \in A_h} \forall \bar{x}. Do(m(\Pi(\bar{x}, s)), s, s') \supset \psi_T;
4. $D_1 \models \forall s. Do(\text{anyllps}, s_0, s) \supset \bigwedge_{A \in A_h} \forall \bar{x}. Do(m(\Pi(\bar{x}, s)), s, s') \supset \psi_T;
5. $D_1 \models \forall s. Do(\text{anyllps}, s_0, s) \supset \xi_P;
6. $D_1 \models \forall s. Do(\text{anyllps}, s_0, s) \supset \xi_f;
7. $D_1 \models \exists h. \text{Infexe}(\text{anyllps}, h, C_1) \equiv \exists h. \text{Infexe}(\text{anyllps}, h, C_h);
8. $D_1 \models \forall s. Do(\text{anyllps}, s_0, s) \land m(G_h)[s] \supset G_1[s].$

**Proof.** Firstly, we prove the only-if direction:

1. For any $M_h^\phi \models D_h^\phi$, we can extend it to a model $M_1$ of $D_1$. Then there exists a model $M_h$ of $D_h$, which is a sound $m$-abstraction of $M_1$, and we have that the initial situation $S_0^{M_1}$ of $M_h$ is $m$-isomorphic to the initial situation $S_0^{M_1}$ of $M_1$. Given $\phi \in D_h^\phi$, since $S_0^{M_1}$ satisfies $\phi$, so $S_0^{M_h}$ satisfies $\phi$. Therefore, $D_h^\phi \models m(\phi)$ for $\phi \in D_h^\phi$.

2. Let $M_l$ be a model of $D_1$, then there exists a model $M_h$ of $D_h$, which is a complete $m$-abstraction of $M_l$ via a $m$-simulation relation $B_1$. Let $s_l$ be a situation of $M_l$ which satisfies $Do(\text{anyllps}, s_0, s)$, then based on the definition of $m$-simulation relation, we have that there is a situation $s_h$ of $M_h$ such that $(s_h, s_l) \in B_1$. Thus, for any HL action $A(\bar{x})$, $s_h$ satisfies $\forall \bar{x}.\Pi(A(\bar{x}), s)$ iff $s_l$ satisfies $\forall \bar{x}.m(\Pi(A(\bar{x}), s))$. Furthermore, if $M_l, v[s/s_l] \models \neg \forall \bar{x}. Do(m(A(\bar{x})), s, s')$, then we have $M_h, v[s/s_h] \models \neg \forall \bar{x}.\exists \bar{u}. Do(m(A(\bar{x})), s, s')$, which implies $M_h, v[s/s_h] \models \forall \bar{x}.\Pi(A(\bar{x}, s))$. Therefore, $M_l, v[s/s_l] \models \forall \bar{x}.m(\Pi(A(\bar{x}, s))$. 

3. Let $M_1$ be a model of $D_1$, then there exists a model $M_h$ of $D_h$ such that $M_h$ is a complete abstraction for $M_1$ via a $m$-simulation relation $B_1$, and sound abstraction for $M_1$ via a $m$-back-simulation relation $B_2$. Let $s_1$ be a situation of $M_1$ which satisfies $Do(\text{angyllps}, S_0, s)$, then based on the definitions of $m$-simulation and $m$-back-simulation relation, we have that there is a situation $s_h$ of $M_h$ such that $(s_h, s_1) \in B_1$, and a situation $s'_1$ of $M_1$ such that $(s_h, s'_1) \in B_2$. Thus, $s_1$ and $s'_1$ are $R$-related. Furthermore, for any HL action $A(\vec{x})$, if $M_1, v[s/s_1] \models \forall \vec{x}.m(P(A(\vec{x}), s))$, then $M_h, v[s/s_h] \models \forall \vec{x}.\Pi(A(\vec{x}), s)$, and hence $M_1, v[s/s'_1] \models \forall \vec{x}.\Pi(A(\vec{x}), s)$. Therefore, we have $M_1, v[s/s_1] \models \forall \vec{x}, s'. R(s, s') \land Do(m(A(\vec{x})), s', s'_1)$. 

4. Let $M_1$ be a model of $D_1$, then there exists a model $M_h$ of $D_h$ such that $M_h$ is a complete abstraction for $M_1$ via a $m$-simulation relation $B_1$. Let $s_1$ be a situation of $M_1$ which satisfies $Do(\text{angyllps}, S_0, s)$, then based on the definition of $m$-simulation relation, we have that there is a situation $s_h$ of $M_h$ such that $(s_h, s_1) \in B_1$, and $M_1, v[s/s_1] \models \forall \vec{x}.h.\text{Term}(A(\vec{x}), s, C_1)$ for any high HL action $A(\vec{x})$.

5. Let $M_1$ be a model of $D_1$, then there exists a model $M_h$ of $D_h$, which is a complete $m$-abstraction of $M_1$ via a $m$-simulation relation $B_1$. Let $s_1$ be a situation of $M_1$ which satisfies $Do(\text{angyllps}, S_0, s)$, then based on the definition of $m$-simulation relation, we have that there is a situation $s_h$ of $M_h$ such that $(s_h, s_1) \in B_1$. For each HL action $A(\vec{x})$, and any situation $s_1$, if $M_1, v[s/s_1, s'/s'_1] \models \forall \vec{x}.Do(m(A(\vec{x})), s, s')$, then there exists an execution $\pi(\vec{x}, \vec{y})$ of $A(\vec{x})$, and a situation $s'_1$ such that $M_h, v[s/s_h, s'/s'_1] \models \forall \vec{x}, \vec{y}.Do(\pi(\vec{x}, \vec{y}), s, s')$, and $(s_h, s'_1) \in B_1$. Then we have that $s_1$ satisfies $\forall \vec{x}, \vec{y}. P(\vec{y}, s')$ iff $s'_1$ satisfies $\forall \vec{y}. P(\vec{y}, s'_1)$.

6. The proof of item 6 is similar to item 5.

7. Let $M_1$ be a model of $D_1$, then there exists a model $M_h$ of $D_h$ such that $M_h$ is a complete abstraction for $M_1$ via a $m$-simulation relation $B_1$, and sound abstraction for $M_1$ via a $m$-back-simulation relation $B_2$.

(i) On one hand, Let $h_1$ be a LL infinite history, for an infinite ground HL action sequence $A_1, A_2, \ldots, A_i, \ldots$, and let $\sigma = A_1, A_2, \ldots, A_i$, we consider the infinite LL situation sequence $S_0^{M_h}, s_1^{M_h}, s_2^{M_h}, \ldots, s_i^{M_h}$, where $s_i^{M_h}$ satisfies the following formula $s_i^{M_h} \models h_{i-1} \land Do(m(\sigma), S_0, s_i^{M_h})$. Based on the $m$-simulation relation $B_1$, we can easily construct an HL infinite history $h_{i-1}$ such that $\{h_{i-1}, h_1\} \in B_1$. Furthermore, we can get that if $h_{i-1}$ satisfies $M_1, v[h/h_1] \models Infexe(\text{angyllps}, h_1, C_1)$, then $h_1$ satisfies $M_h, v[h/h_1] \models Infexe(\text{angyllps}, h_1, C_1)$. Then, based on the Proposition 4, we can get that $M_1, v[h/h_1] \models Infexe(\text{angyllps}, h_1, m(C_h))$.

(ii) On the other hand, given an LL infinite history $h_{i-1}$, we can also construct an HL infinite history $h_{i-1}$ based on the $m$-simulation relation $B_1$ such that such that $\{h_{i-1}, h_1\} \in B_1$. If $h_1$ satisfies $M_1, v[h/h_1] \models Infexe(\text{angyllps}, h_1, m(C_h))$, then according to the proposition 4, we can get that $M_h, v[h/h_1] \models Infexe(\text{angyllps}, h_1, C_1)$. Given the definition of $m$-back-simulation relation, we then have that there exists a low-level infinite history $h_1$ which satisfies $M_1, v[h/h_1] \models Infexe(\text{angyllps}, h_1, C_1)$.

8. Let $M_1$ be a model of $D_1$, then there exists a model $M_h$ of $D_h$, which is a complete $m$-abstraction of $M_1$ via a $m$-simulation relation $B_1$. Let $s_1$ be a situation of $M_1$ which satisfies $Do(\text{angyllps}, S_0, s)$, then based on the definition of $m$-simulation relation, we have that there is a situation $s_h$ of $M_h$, such that $(s_h, s_1) \in B_1$. Suppose $s_1$ satisfies $m(G_h)$, then $s_h$ satisfies $G_h$, according to the definition of weak sound abstraction on theory level, we can get that $s_1$ satisfies $G_1$.

Secondly, we prove the if direction: For any model $M_1$ of $D_1$, we construct a model $M_h$ of $D_h$ as follows: $M_h$ interprets the HL relational and functional fluents at initial situation $S_0^{M_1}$ according to the initial situation $S_0^{M_h}$ of $M_h$ and the refinement mapping. We complete $M_h$ by using the action precondition and effect axioms.

For $M_1$ and $M_h$, we firstly prove that $M_h$ is a complete abstraction of $M_1$: In the following, we construct a $m$-simulation relation $B_1$ between the situation and infinite history domains of $M_h$ and $M_1$.

1. $(S_0^{M_h}, S_0^{M_1}) \in B_1$, i.e., $S_0^{M_h}$ is $m$-isomorphic to $S_0^{M_1}$.

2. Let $(s_h, s_1) \in B_1$, where $s_1$ is a situation reached from $S_0^{M_1}$ via executing anyllps, then $s_h$ is $m$-isomorphic to $s_1$. Let $A(\vec{x})$ be an arbitrary HL action. By item 4, if $m(A(\vec{x}))$ is executable in the situation $s_1$, then we have $M_1, v[s/s_1] \models \forall \vec{x}.h.\text{Term}(m(A(\vec{x})), s, C_1)$. By item 2, if $m(A(\vec{x}))$ is executable in $s_h$, then $A(\vec{x})$ is executable in $s_h$. For each situation $s'_1$ satisfies $Do(m(A(\vec{x})), s_1, s'_1)$ and all the situations $s'_1$ that satisfy $Do(m(A(\vec{x}), s_1, s'_1))$, we add all the pairs $(s'_h, s'_1) \in B_1$. By item 5 (the case of item 6 can be discussed similarly), for an execution $A_h(\vec{x}, \vec{u})$ of $A(\vec{x})$, $s'_1$ satisfies $\forall \vec{y}. m(P(\vec{y}, s'_1))$ iff $s'_1$ satisfies $\forall \vec{y}. P(\vec{y}, s'_1)$ for the situations $s'_1$ that do not satisfy $\forall \vec{y}. P(\vec{y}, s'_1)$, we then delete all the pairs $(s'_h, s'_1)$ from $B_1$.

3. $B_1$ is a $m$-simulation relation follows its construction coupled with item 7.

Furthermore, if $(s_h, s_1) \in B_1$ and $s_h$ satisfies $G_h$, then $s_1$ satisfies $G_1$. Since $s_h$ and $s_1$ are $m$-isomorphic, $s_1$ satisfies $m(G_h)$, by Item 8, $s_1$ satisfies $G_1$. Therefore, $M_h$ is a sound abstraction of $M_1$.

For $M_1$ and $M_h$, we then prove that $M_h$ is a sound abstraction of $M_1$: In the following, we construct a $m$-back-simulation relation $B_2$ between the situation and infinite history domains of $M_h$ and $M_1$,
1. \( S_0^{M_h}, S_0^{M_l} \) ∈ \( B_2 \), i.e., \( S_0^{M_l} \) is \( m \)-isomorphic to \( S_0^{M_h} \).

2. Let \( \langle s_h, s_l \rangle \in B_2 \), where \( s_l \) is a situation reached from \( S_0^{M_l} \) via executing anyllps, then \( s_h \) is \( m \)-isomorphic to \( s_l \). Let \( A(\vec{x}) \) be an arbitrary HL action. By item 4, if \( m(A(\vec{x})) \) is executable in the situation \( s_l \), then we have \( M_l, v[s_l/s] \triangleright \forall \vec{x}.\text{Term}(m(A(\vec{x})), s, C_l) \). By item 3, if \( s_l \) satisfies \( \forall \vec{x}.m(\Pi_A(\vec{x}, s_l)) \), then there exists a situation \( s'_l \) \( R \)-related to \( s_l \) satisfies \( \forall \vec{x}, \exists s''.\text{Do}(m(A(\vec{x})), s', s'') \). Then \( s_l \) and \( s'_l \) are \( m \)-isomorphic, and we replace \( \langle s_h, s_l \rangle \) in \( B_2 \) by \( \langle s_h, s'_l \rangle \). In addition, for each situation \( s'_l \) satisfies \( \vec{x}.\text{Do}(m(A(\vec{x})), s_l, s'_l) \) and the all the situations \( s''_l \) that satisfy \( \vec{x}.\text{Do}(m(A(\vec{x})), s_l, s''_l) \), we add all the pairs \( \langle s'_h, s''_l \rangle \) in \( B_2 \). By item 5 (the case of item 6 can be discussed similarly), for any execution \( A_d(\vec{x}, \vec{u}) \) of \( A(\vec{x}) \), \( s''_l \) satisfies \( \forall \vec{y}, m(P(\vec{y}, s''_l)) \) iff \( s_l \) satisfies \( \forall \vec{y}, \forall \vec{u}.m(\Pi_A(\vec{x}, \vec{y}, \vec{u}, s_l)) \) iff \( s_l \) satisfies \( \forall \vec{y}, \forall \vec{u}.m(\Pi_A(\vec{x}, \vec{y}, \vec{u}, s_l)) \). The situations \( s'_l \) that do not satisfy \( \forall \vec{y}.P(\vec{y}, s'_l) \), we then delete all the pairs \( \langle s'_h, s''_l \rangle \) from \( B_2 \).

3. \( B_2 \) is \( m \)-back-simulation relation follows its construction coupled with item 7.

Furthermore, if \( \langle s_h, s_l \rangle \in B_2 \) and \( s_h \) satisfies \( G_h \), then \( s_l \) satisfies \( G_l \). Since \( s_h \) and \( s_l \) are \( m \)-isomorphic, \( s_l \) satisfies \( m(G_h) \), by Item 8, \( s_l \) satisfies \( G_l \).

In conclusion, according to the definition of sound abstraction on theory level, we can get that \( G_h \) is a sound abstraction of \( G_l \).

Unfortunately, we are not able to give a proof-theoretic characterization for complete abstractions. Nonetheless, we give the following characterization where Item 2 says that \( M_1 \) satisfies for any executable refinement of a HL situation, \( \psi_T \), \( \xi_P \), and \( \xi_l \) hold, and the HL goal implies the matched HL goal, and the executability of the refinement of any HL action implies its mapped precondition; Item 3 says that \( M_2 \) satisfies that there exists a set \( P \) of situations including the initial situation such that for any \( P \)-situation, \( \psi_T \), \( \xi_P \), and \( \xi_l \) hold, and the HL goal implies the matched HL goal, and the mapped precondition of any HL action implies that its refinement is executable and leads to a \( P \)-situation.

**Theorem 2.** \( G_h \) is a complete \( m \)-abstraction of \( G_l \) iff for any model \( M_h \) of \( \mathcal{D}_h \), there exists a model \( M_2 \) of \( \mathcal{D}_l \) such that:

1. \( S_0^{M_h} \sim_m S_0^{M_l} \).

2. \( M_1 \models \forall s.\text{Do}(\text{anyllps}, S_0, s) \supset \forall A \in A_h \forall \vec{x}, \vec{s}'.\text{Do}(m(A(\vec{x})), s, s') \supset m(\Pi_A(\vec{x}, s)) \).

3. \( M_1 \models \forall s.\text{Do}(\text{anyllps}, S_0, s) \supset \forall A \in A_h \forall \vec{x}, \vec{s}'.\text{Do}(m(A(\vec{x})), s, s') \supset \forall A \in A_h \forall \vec{x}.\text{Term}(m(A(\vec{x})), s, C_l) \).

4. \( M_1 \models \forall s.\text{Do}(\text{anyllps}, S_0, s) \supset \xi_P \).

5. \( M_1 \models \forall s.\text{Do}(\text{anyllps}, S_0, s) \supset \xi_l \).

6. \( M_1 \models \forall s.\text{Do}(\text{anyllps}, S_0, s) \supset m(G_h) \).

7. if \( M_1 \models \exists h_i.\text{Infexe}(\text{anyllps}, h_i, C_l) \), then \( M_h \models \exists h_i.\text{Infexe}(\text{anyllps}, h_i, C_h) \).

and there is another model \( M_2 \) of \( \mathcal{D}_l \) and a situation set \( P \) of \( M_2 \) such that:

8. \( S_0^{M_h} \sim_m S_0^{M_2} \).

9. \( M_2 \models P(S_0) \land \forall s, P(s) \supset \forall A \in A_h \forall \vec{x}, \exists s'.\text{Do}(m(A(\vec{x})), s, s') \land P(s') \).

10. \( M_2 \models P(S_0) \land \forall s, P(s) \supset \forall A \in A_h \forall \vec{x}, \exists s'.\text{Do}(m(A(\vec{x})), s, s') \land P(s') \supset \psi_T \).

11. \( M_2 \models P(S_0) \land \forall s, P(s) \supset \xi_P \).

12. \( M_2 \models P(S_0) \land \forall s, P(s) \supset \xi_l \).

13. \( M_2 \models P(S_0) \land \forall s, P(s) \supset G_l[s] \supset m(G_h)[s] \).

14. if \( M_h \models \exists h_i.\text{Infexe}(\text{anyllps}, h_i, C_h) \), then \( M_2 \models \exists h_i.\text{Infexe}(\text{anyllps}, h_i, C_l) \).

**Proof.** Firstly, we prove the **only-if** direction:

1. According to the definition of complete abstraction on theory level, we know that for each high-level model \( M_h \) of \( \mathcal{D}_h \), there exist a LL model \( M_1 \) such that \( M_h \) is a complete abstraction of \( M_1 \) via a \( m \)-simulation relation \( B_1 \). Then, we have \( S_0^{M_h} \sim_m S_0^{M_1} \).

2. Let \( s_l \) be a situation of \( M_1 \) which satisfies the formula \( \text{Do}(\text{anyllps}, S_0, s) \), then based on the definition of \( m \)-simulation relation, we have that there is a situation \( s_h \) of \( M_h \) such that \( \langle s_h, s_l \rangle \in B_1 \). Thus, for any HL action \( A(\vec{x}) \), \( s_h \) satisfies \( \forall \vec{x}.\text{Term}(A(\vec{x}), s, C_l) \). Furthermore, if \( M_1 \models \forall s, \forall s' \exists s'.\text{Do}(m(A(\vec{x})), s, s') \), then \( M_h \models \forall s, \forall s' \exists s'.\text{Do}(A(\vec{x}), s, s') \), which means that \( M_h \models \forall s, \forall s' \exists s'.\text{Do}(A(\vec{x}), s, s') \). Therefore, \( M_1 \models \forall s, \forall s' \exists s'.\text{Do}(A(\vec{x}), s, s') \).

3. Let \( s_l \) be a situation of \( M_1 \) which satisfies the formula \( \text{Do}(\text{anyllps}, S_0, s) \), then based on the definition of \( m \)-simulation relation, we have that there is a situation \( s_h \) of \( M_h \) such that \( \langle s_h, s_l \rangle \in B_1 \), and \( M_1 \models \forall s, \forall s' \exists s'.\text{Do}(A(\vec{x}), s, s') \). Thus, \( M_h \models \forall s, \forall s' \exists s'.\text{Do}(A(\vec{x}), s, s') \).

4. Let \( s_l \) be a situation of \( M_1 \) which satisfies the formula \( \text{Do}(\text{anyllps}, S_0, s) \), then based on the definition of \( m \)-simulation relation, we have that there is a situation \( s_h \) of \( M_h \) such that \( \langle s_h, s_l \rangle \in B_1 \), and \( M_1 \models \forall s, \forall s' \exists s'.\text{Do}(A(\vec{x}), s, s') \). Thus, \( M_h \models \forall s, \forall s' \exists s'.\text{Do}(A(\vec{x}), s, s') \).

5. The proof of item 5 is similar to item 4.

6. Let \( s_l \) be a situation of \( M_1 \) which satisfies the formula \( \text{Do}(\text{anyllps}, S_0, s) \), then based on the definition of \( m \)-simulation relation, we have that there is a situation \( s_h \) of \( M_h \) such that \( \langle s_h, s_l \rangle \in B_1 \). Suppose \( s_l \) satisfies
$G_l$, then $s_h$ satisfies $G_h$, based on the weak complete abstraction on theory level. Thus, we have that $s_t$ satisfies $m(G_h)$.

7. According to the definitions of complete abstraction and $m$-simulation, if there exists a LL infinite history $h_t$ such that $M^l_1, v[h/h_t] \models \text{Infexe}(\text{anyllps}, h, C_l)$, we can construct an HL infinite history $h_h$ such that $\langle h_h, h_t \rangle \in B_1$. Then we can get $M_h, v[h/h_h] \models \text{Infexe}(\text{anyllps}, h, C_h)$.

For item 8-14, we firstly construct a situation set $P$ of $M^l_2$ as follows: (i) $S^M_0 \subseteq P$; (ii) for any LL program anyllps, if there is a situation $s_t$ which satisfies $\text{Do}(\text{anyllps}, S^M_0, s_t)$, then we let $s_t \in P$. Now we prove item 8-14.

8. According to the definition of complete abstraction on theory level, we know that for each high-level model $M_h$ of $D_h$, there exist a LL model $M^l_2$, such that $M_h$ is a sound abstraction of $M^l_2$ of via a $m$-back-simulation relation $B_2$. Then, we have $S^M_0 \sim_m S^M_1$.

9. Let $s_h$ be a reachable situation of $M_h$ via executing anyllps, then according to the construction of $P$, we can find a situation $s_t \in P$ of $M^l_2$, such that $\langle s_h, s_t \rangle \in B_2$. Furthermore, for any HL action $A(x)$, we have that if $s_h$ satisfies $\exists \overrightarrow{x}.\Pi_\mathcal{A}(A(x), s)$, then there exists a situation $s_t'$ such that $s_t$ satisfies $\exists \overrightarrow{x}.\text{Do}(m(A(x)), s_t, s_t')$ and $s_t' \in P$.

10. Let $s_h$ be a reachable situation of $M_h$ via executing anyllps, then according to the construction of $P$, we have that there exists a situation $s_t \in P$ of $M^l_2$, such that $\langle s_h, s_t \rangle \in B_2$. Then for any HL action $A(x)$, based on the definition of $m$-back-simulation, we can get that $M^l_2, v[s/s_t] \models \exists \overrightarrow{x}.\text{Term}(A(x), s, C_l)$.

11. Let $s_h$ be a reachable situation of $M_h$ via executing anyllps, then according to the construction of $P$, we have that there exists a situation $s_t \in P$ of $M^l_2$, such that $\langle s_h, s_t \rangle \in B_2$. For each HL action $A(x)$, if there exists an execution $A(x, \overrightarrow{u})$ of $A(x)$, and a situation $s_h'$, such that $M_h, v[s/s_h, s'/s_h'] \models \exists \overrightarrow{x}.\exists \overrightarrow{u}.\text{Do}(A_d(x, \overrightarrow{u}), s, s')$, then based on the definition of $m$-back-simulation and the construction of the $P$, we can get that there exists a situation $s_t'$ such that $M^l_2, v[s/s_t, s'/s_t'] \models \exists \overrightarrow{x}.\text{Do}(m(A(x)), s_t, s_t') \wedge P(s')$, and $\langle s_h', s_t' \rangle \in B_2$. Then, $s_t$ satisfies $\exists \overrightarrow{x}.\exists \overrightarrow{u}.\text{Do}(A_d(x, \overrightarrow{u}, s, s))$ iff $s_h'$ satisfies $\exists \overrightarrow{x}.\exists \overrightarrow{u}.\text{Do}(A_d(x, \overrightarrow{u}, s, s))$ iff $s_t'$ satisfies $\exists \overrightarrow{y}.P(\overrightarrow{y}, s')$ iff $s_t'$ satisfies $\exists \overrightarrow{y}.m(P(\overrightarrow{y}, s'))$.

12. The proof of item 12 is similar to item 11.

13. Let $s_h$ be a reachable situation of $M_h$ via executing anyllps, then based on the construction of $P$, there is a situation $s_t$ of $M^l_2$, such that $\langle s_h, s_t \rangle \in B_2$. Suppose $s_t$ satisfies $G_t$, then according to the condition 2 of the definition of complete abstraction on theory level, we have that $s_h$ satisfies $G_h$. Thus, $s_t$ satisfies $m(G_h)$.

14. According to the definitions of sound abstraction and $m$-back-simulation, if there exists a HL infinite history $h_h$ such that $M_h, v[h/h_h] \models \text{Infexe}(\text{anyllps}, h, C_h)$, then we can construct an LL infinite history $h_t$ such that $\langle h_h, h_t \rangle \in B_2$. Then we can get that $M^l_2, v[h/h_t] \models \text{Infexe}(\text{anyllps}, h, C_t)$.

Secondly, we prove the if direction:

For $M^l_1$ and $M_h$, we firstly prove that $M_h$ is a complete abstraction of $M^l_1$. For this purpose, we construct a $m$-simulation relation $B_3$ between the situation and infinite history domains of $M_h$ and $M^l_1$ as follows:

1. $\langle S^M_0, S^M_1 \rangle \in B_1$, i.e., $S^M_0$ is $m$-isomorphic to $S^M_1$.

2. Let $\langle s_h, s_t \rangle \in B_1$, where $s_t$ is a situation reached from $\langle s_h, s_t \rangle$ via executing anyllps, then $s_h$ is $m$-simulation to $s_t$. Let $A(x)$ be an arbitrary HL action. By item 3, if $m(A(x))$ is executable in the situation $s_t$, then we have $M_1, v[s/s_t] \models \forall \overrightarrow{x}.\text{Term}(m(A(x)), s, C_t)$. By item 2, if $m(A(x))$ is executable in $s_t$, then $A(x)$ is executable in $s_h$. For each situation $s'_h$ satisfies $\forall \overrightarrow{x}.\text{Do}(m(A(x)), s'_h, s'_h)$ and all the situations $s'_h$ satisfies $\forall \overrightarrow{x}.\text{Do}(A(x), s'_h, s'_h)$, we add all the pairs $\langle s'_h, s'_h \rangle$ in $B_3$. By item 4 (the case of item 5 can be discussed similarly), for any execution $A_d(x, \overrightarrow{u})$ of $A(x)$, $s'_h$ satisfies $\forall \overrightarrow{x}.m(P(\overrightarrow{y}, s'_h))$ iff $s'_h$ satisfies $\forall \overrightarrow{x}.\exists \overrightarrow{u}.\text{Do}(A(x), \overrightarrow{u}, s'_h)$ iff $s'_h$ satisfies $\forall \overrightarrow{y}.\exists \overrightarrow{u}.\text{Do}(A_d(x, \overrightarrow{u}, s'_h))$ iff $s'_h$ satisfies $\forall \overrightarrow{y}.\exists \overrightarrow{u}.\text{Term}(m(A(x)), \overrightarrow{u}, s'_h)$. For the situations $s'_h$ that do not satisfy $\forall \overrightarrow{y}.P(\overrightarrow{y}, s'_h)$, we then delete all the pairs $\langle s'_h, s'_h \rangle$ from $B_3$.

3. $B$ is a $m$-simulation relation follows its construction coupled with item 7.

Furthermore, if $\langle s_h, s_t \rangle \in B_1$ and $s_t$ satisfies $G_t$, then $s_h$ satisfies $G_h$. Since $s_h$ and $s_t$ are $m$-isomorphic, $s_t$ satisfies $G_t$, by item 6, $s_t$ satisfies $m(G_h)$, thus $s_h$ satisfies $G_h$.

For $M^l_2$ and $M_h$, we prove that $M_h$ is a sound abstraction of $M^l_2$. For this purpose, in the following, we construct a $m$-back-simulation relation $B_4$ between the situation and infinite history domains of $M_h$ and $M^l_2$. Particularly, we consider the situation set $P$ of $M^l_2$ we constructed above.

1. $\langle S^M_0, S^M_2 \rangle \in B_2$, i.e., $S^M_0$ is $m$-isomorphic to $S^M_2$, where $S^M_2 \subseteq P$.

2. Let $\langle s_h, s_t \rangle \in B_2$, where $s_t$ is a situation reached from $\langle s_h, s_t \rangle$ via executing anyllps, then $s_h$ is $m$-simulation to $s_t$. Let $A(x)$ be an arbitrary HL action. By item 10, if $m(A(x))$ is executable in the situation $s_t$, then we have $M_1, v[s/s_t] \models \text{Term}(m(A(x)), s, C_t)$. By item 9, if $m(A(x))$ is executable in $s_t$, then $M_1, v[s/s_t] \models \text{Term}(m(A(x)), s, C_t)$. For each situation $s'_h$ satisfies $\forall \overrightarrow{x}.\text{Do}(A(x), s'_h, s'_h)$ and all the situations $s'_h$ satisfies $\forall \overrightarrow{x}.\text{Do}(A(x), s'_h, s'_h)$, we add all the pairs $\langle s'_h, s'_h \rangle$ in $B_2$. By item 11 (the case of item 12 can be discussed similarly), for any execution $A_d(x, \overrightarrow{u})$ of $A(x)$, $s'_h$ satisfies $\forall \overrightarrow{x}.\exists \overrightarrow{u}.\text{Do}(A(x), \overrightarrow{u}, s'_h)$ iff $s'_h$ satisfies $\forall \overrightarrow{x}.\exists \overrightarrow{u}.\text{Do}(A_d(x, \overrightarrow{u}, s'_h))$ iff $s'_h$ satisfies $\forall \overrightarrow{x}.\exists \overrightarrow{u}.\text{Term}(m(A(x)), \overrightarrow{u}, s'_h)$. For the situations $s'_h$ that do not satisfy $\forall \overrightarrow{y}.P(\overrightarrow{y}, s'_h)$, we then delete all the pairs $\langle s'_h, s'_h \rangle$ from $B_2$. 

3. $B_2$ is a $m$-back-simulation relation follows its construction coupled with item 14.

Furthermore, if $(s_h, s_t) \in B_2$ and $s_t$ satisfies $G_t$, then $s_h$ satisfies $G_h$. Since $s_h$ and $s_t$ are $m$-isomorphic, $s_t$ satisfies $G_t$, by Item 13, $s_l$ satisfies $m(G_h)$, thus $s_h$ satisfies $G_h$.

\[ \square \]

4 Implementation Theory

In this section, by introducing some restrictions in Theorem 1, we first give a sufficient condition for sound abstractions that is first-order verifiable. Then we discuss how to verify the sufficient condition with first-order theorem prover when the g-planning problem abstractions are QNPs.

We first discuss how to obtained a sufficient condition for sound abstractions from Theorem 1. In the following, we use $\models_f$ to denote classic first-order entailment:

1. Situations that satisfy $\text{Do(angel lips, } S_0, s\) are all executable. State constraints are formulas that hold in all executable situations. Given a BAT $D$ and a formula $\phi(s)$, $\phi(s)$ is a state constraint for $D$ if $D \models \forall s.\text{Exec}(s) \supset \phi(s)$. Thus, for getting a sufficient condition for sound abstractions, we replace the condition $\text{Do(angel lips, } S_0, s\)$ by providing LL state constraints in tasks 2, 3, 4 and 6 in Theorem 1. We use $D_{sc}$ to denote a set of state constraints, and abuse $D_{sc}$ as the conjunction of its elements.

(2). Determining whether a given program terminates is the quintessential undesirable problem. We assume that any HL action refinement does not involve iteration. Then, the formula $\psi_T$ in task 4 is true trivially.

(3). Since each HL action is deterministic, thus, we can ignore trajectory constraints, and then ignore task 5.

(4). Since task 2 involves the reasoning about LL programs, we use existentially extended regression to compute executability conditions of Golog programs. Given a program $\delta$ and a situation $s$, the executability condition $\text{pre}((\delta, s))$ of $\delta$ in the situation $s$ can be computed as $R^E([s], A(\vec{x}))$.

For space saving, we use the following abbreviation:

$$\xi_A \doteq \land_{A \in A_h} \forall \vec{x}. \text{pre}(m(A(x)), s) \equiv m(\Pi_A(x(s)),$$

Again, for getting a sufficient condition for sound abstractions, we verify the following stronger task which can implies both task 2 and task 3:

$$\models_f \forall s.\text{D}_{sc}(s) \supset \xi_A$$

(5). For further discussion, we introduce two more abbreviations as follows:

$$\zeta_P \doteq \land_{A \in A_h} \forall \vec{x}. \text{pre}(m(A(x)), s)$$

$$\supset \lor_{P \in P_h} \forall \tilde{y}. [R^E[m(P(\tilde{y}))[s], m(A(\vec{x}))]

$$\supset m(\phi_P(\vec{x}, y, s)) \lor m(\phi_P(x, y, s))

$$\lor R^U[m(P(\tilde{y}))[s], m(A(\vec{x}))].$$

$$\zeta_f \doteq \land_{A \in A_h} \forall \vec{x}. \text{pre}(m(A(x)), s)$$

$$\supset \lor_{f \in F_h} \forall y, z. [R^E[m(f(y, z))[s], m(A(\vec{x}))]

$$\supset m(\psi_f(A(x, y, z, s)) \lor m(\psi_f(A(x, y, z)))

$$\lor R^U[m(f(y, z))[s], m(A(\vec{x}))].$$

Based on (1) and the two extended regression definitions, we can get that task 4 is equivalent to:

$$\models_f \forall s.\text{D}_{sc}(s) \supset \zeta_P \land \zeta_f.$$

We have the following result based on the analysis above:

**Corollary 1.** Given a $g$-planning problem $G_0$ and its abstraction $G_h$, suppose that all the HL actions are deterministic and their refinements do not involve iteration, then $G_h$ is a sound abstraction of $G_0$ if:

1. $D_{sc}^G \models f_o m(\phi)$, where $\phi \in D_{sc}^G$;
2. $\models_f \forall s.\text{D}_{sc}(s) \supset \xi_A$;
3. $\models_f \forall s.\text{D}_{sc}(s) \supset \phi_P \land \zeta_f$;
4. $\models_f \forall s.\text{D}_{sc}(s) \land m(G_h)[s] \supset G_l[s].$

The soundness of abstractions can be verified by feeding all 4 tasks above into a theorem prover. However, existing first-order theorem provers do not support counting or transitive closure. QNPs are popular abstraction models for g-planning problems. We now turn to discuss how to verify QNP abstractions. In particular, we develop methods to handle the counting and transitive closure.

QNPs are extensions of classical planning problems. Compared to classical planning problems, state variables of QNPs contain non-negative numerical variables. These variables introduce the non-propositional atoms $n = 0$ and their negations $n > 0$. These literals can appear in the initial situations, action preconditions, and goals of QNPs. The effects of actions of QNPs on a numerical variable $n$ can be increments or decrements denoted by $n \uparrow$ and $n \downarrow$, respectively.

**Example 1 cont’d.** An abstraction for the g-planning problem $Clear A$ is a QNP $Q = \langle \{F, V, I, O, G\},$ where $F = \{H\}$ contains a propositional variable $H$ that means the agent holding a block, $V = \{n\}$ contains a numerical variable $n$ that means the number of blocks above $A$. The initial state $I$ is $n > 0 \land \neg H$, the goal state $G$ is $n = 0$, and the actions $O = \{\text{pick above, putaside}\}$ are defined as follows:

pick above : $\langle \neg H \land n > 0; H, n \downarrow \rangle$; put aside : $\langle H; \neg H \rangle$.

In QNP abstraction case, the verification for task 3 in Theorem 1 can be more specific. For each HL relational fluent $p \in P_h$, and HL action $A \in A_h$, $A$ makes $p$ true (the false case can be discussed similarly) means that in LL, all the executions of $m(A)$ make $p$ true, then we have the task $3^*$:

$$\models_f \forall s.\text{D}_{sc}(s) \land \forall \vec{x}. \text{pre}(m(A(x)), s) \supset R^E[m(p(s)), m(A(\vec{x}))].$$

For the verification tasks involving functional fluents, we only discuss a restricted case, that is, effects of actions on functional fluents are $\pm 1$. Given a functional fluent $f \in F_h$, and action $A \in A_h$, if $A$ makes $n$ increase by 1 (the decrease case can be discussed similarly), we prove the task $4^*$:

$$\models_f \forall s.\text{D}_{sc}(s) \land \forall \vec{x}. \text{pre}(m(A(x)), s) \supset \forall s'. \text{Do}(m(A(x)), s, s') \supset$$

$$\supset \forall y, z, m(f(y, z), k + 1) \equiv m(\psi_f(A(x, y), g, k)).$$

which means that if $m(A)$ is executable in any LL situation $s$, then for any situations $s'$ that can arrived by executing $m(A)$ from $s$, we have $m(n)[s'] = m(n)[s] + 1$. Assuming that $m(n) = \# x. \phi(x)$, we have the formula $\psi$ below that holds:

$$\exists x. \phi(x, s') \land \neg \phi(x, s) \land \forall x. \phi(x, s) \supset \phi(x, s') \land\$$

$$\forall x, y. \phi(x, s) \land \neg \phi(x, s') \land \phi(y, s') \land \phi(y, s) \supset x = y.$$

This formula says that: there exists one and only one object that makes $\phi(x)$ true from false after a program execution.
Then base on Proposition 3, we can get the following task 4#, which is equivalent to task 4*:
\[
\exists \vec{x} \forall s. D_{sc}(s) \land \\
\forall \vec{x}. \text{pre}(m(A(\vec{x})), s) \supset R^U[\Psi(s), m(A(\vec{x}))].
\]

Transitive closure formulas are often used to define counting terms. Thus, task 4# may involve regression about transitive closure. In fact, the definition of one-step regression of transitive closure formulas is the same as Definition 1.

Example 1 cont’d. Given the successor state axiom of fluent \( on(x, y, s) \) in Example 1, and a transitive closure formula \( [TC_{x, y} on(x, y, do(unstack(A, B), s))](x, C) \), the regression result \( R_P[\phi] \) of \( \phi \) related to the concrete action \( unstack(A, B) \) is as follows:
\[
[TC_{x, y} on(x, y, s) \land (x \neq A \lor y \neq B)](x, C).
\]

To handle the regression of transitive closure formulas with existing theorem provers, for a given transitive closure formula \( \phi(\vec{x}) \), we first get the regression result \( R_P[\phi(\vec{x})] \) for \( \phi(\vec{x}) \). Then we define \( R_P[\phi(\vec{x})] \) as a new relation \( P(\vec{x}) \) and endow it with the transitivity and the minimality.

5 Verification System and Experiment Results

Based on the discussion in Section 4, we designed a sound abstraction verification system for g-planning. The inputs of our system include a g-planning problem coupled with state constraints, a refinement mapping, and an abstraction problem. The output of our system is True, False, or Unknown. g-planning problems in our system take the form of STRIPS-like problems. The only extension of g-planning problems to STRIPS problems is that their initial states can be first-order formulas with transitive closure (see Example 1). The formalization of QNP abstractions in our system is the same as that in [Bonet and Geffner, 2020].

The workflow of our verification system is as follows:

Step 1: Given the input g-planning problem, the system automatically generates the LL BAT;

Step 2: Based on the input abstraction problem, refinement mapping, the generated LL BAT, and LL state constraints, the system generates the verification tasks that we mention in Section 4. Concretely, the system generates task 1 for the HL and the LL initial states; task 2 for each HL action; task 3* for each HL relational fluent; task 4# for each HL functional fluent; and task 5 for the HL and the LL goal states.

Step 3: The system feeds all the tasks above into the Z3-solver (version 4.8.10.0) [de Moura and Bjørner, 2008] for verification. If all these tasks can be verified, then the system returns True, else returns False. If the theorem prover times out, the system returns Unknown.

Example 1 cont’d. The refinement mapping about the HL relational \( H \) and functional fluent \( n \) are as follows:

\[
m(H) = \exists x. holding(x); m(n) = \#x.on^+(x, A).
\]

The task 1 generated by our system is as follows:

\[
\notexists x. on^+(x, A) \land \notontable(A) \land \neg holding(x) \supset \notexists x. holding(x) \land \notexists x. on^+(x, A).
\]

Our verification system was tested on 7 domains: ClearA is our running example; OnAB is about achieving the goal \( On(A, B) \) in Blocksworld instances where the gripper is initially empty, and the blocks \( A \) and \( B \) are in different towers with blocks above them; Logistics involves a vehicle whose goal is to load goods from the original location and transport them to the target location; Gripper involves a robot with two grippers whose goal is to move all balls from room A to B; GetLast and FindA are both linked list domain problems. The predicate and the action sets of these two problems are the same. The goal of GetLast is to traverse all the elements in a linked list, while FindA aims at finding the element A in a linked list; Corner contains instances that an agent needs to navigate in a rectangle grid and arrive at the point (0, 0) from any other points \((x, y)\). Abstractions of the problems ClearA, Gripper, and OnAB come from [Bonet et al., 2019a], and abstractions of the problems Logistic, GetLast, FindA, Corner are all provided by hand.

Our experiments were run on a Windows machine with a 3.7GHz CPU and 16GB RAM, the default time limit of each subtask was 10s. We summarize the experimental results in Table 1. \#A is the number of HL actions. \#F is the number of HL functional fluent \( F \). \#P is the number of HL relational fluent \( P \). \( T \) shows the total time costs of all verification tasks. The results show that all abstractions are sound.

| Domain       | #A | #F | #P | T(s) | Result |
|--------------|----|----|----|------|--------|
| ClearA       | 2  | 1  | 2  | 4.1687 | True   |
| Gripper      | 4  | 5  | 2  | 7.0014 | True   |
| Logistics    | 4  | 3  | 2  | 6.0052 | True   |
| OnAB         | 4  | 3  | 8  | 10.2191| True   |
| GetLast      | 2  | 1  | 1  | 3.5302 | True   |
| FindA        | 2  | 1  | 1  | 3.5369 | True   |
| Corner       | 2  | 2  | 0  | 3.6378 | True   |

6 Conclusion

In g-planning, solutions of sound abstractions are those with correctness guarantees for the original problems. In this paper, based on Cui et al.’s work, we explored automatic verification of sound abstractions for g-planning. We gave a proof-theoretic characterization of sound abstractions for g-planning in the situation calculus. Then, we got a sufficient condition for sound abstractions which is first-order verifiable. To implement it, we exploited regression extensions and presented methods to handle counting and transitive closure. In the future, we are interested in automatic verification concerning trajectory constraints for non-deterministic abstractions, such as FOND. We are also interested in automatic learning abstractions and abstraction revision based on the verification of sound abstraction.

Acknowledgments

We acknowledge support from the Natural Science Foundation of China under Grant No. 62076261.
References

[Banihashemi et al., 2017] Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. Abstraction in situation calculus action theories. In Proceedings of the 31st AAAI Conference on Artificial Intelligence, AAAI, 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, IJCAI, pages 4667–4673, 2018.

[Bonet and Geffner, 2020] Blai Bonet and Hector Geffner. Qualitative numeric planning: Reductions and complexity. J. Artif. Intell. Res., 69:923–961, 2020.

[Bonet et al., 2019a] Blai Bonet, Guillem Francès, and Hector Geffner. Learning features and abstract actions for computing generalized plans. In Proceedings of The 33rd AAAI Conference on Artificial Intelligence, AAAI, pages 2703–2710, 2019.

[Bonet et al., 2019b] Blai Bonet, Raquel Fuentetaja, Yolanda E-Martín, and Daniel Borrajo. Guarantees for sound abstractions for generalized planning. In Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI, pages 1566–1573, 2019.

[Cui et al., 2021] Zhenhe Cui, Yongmei Liu, and Kailun Luo. A uniform abstraction framework for generalized planning. In Proceedings of the 30th International Joint Conference on Artificial Intelligence, IJCAI, pages 1837–1844, 2021.

[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.

[de Moura and Björner, 2008] Leonardo Mendonça de Moura and Nikolaj Björner. Z3: an efficient SMT solver. In Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS, pages 337–340, 2008.

[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, IJCAI, 2011.

[Hu and Levesque, 2010] Yuxiao Hu and Hector J. Levesque. A correctness result for reasoning about one-dimensional planning problems. In Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning, KR, pages 362–371, 2010.

[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, AAAI, pages 7610–7618, 2019.

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

[Kuske and Schweikardt, 2017] Dietrich Kuske and Nicole Schweikardt. First-order logic with counting. In Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS, 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, 2015] Naiqi Li and Yongmei Liu. Automatic verification of partial correctness of golog programs. In Proceedings of the 24th International Joint Conference on Artificial Intelligence, IJCAI, pages 3113–3119, 2015.

[Luo et al., 2020] Kailun Luo, Yongmei Liu, Yves Lespérance, and Ziliang Lin. Agent abstraction via forgetting in the situation calculus. In Proceedings of the 24th European Conference on Artificial Intelligence, ECAI, pages 809–816, Santiago de Compostela, August–September 2020. IOS Press.

[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 et al., 2019] Javier Segovia, Sergio Jiménez, and Anders Jonsson. Computing programs for generalized planning using a classical planner. Artif. Intell., 272:52–85, 2019.

[Srivastava et al., 2008] Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. Learning generalized plans using abstract counting. In Proceedings of the 23rd AAAI Conference on Artificial Intelligence, AAAI, pages 991–997, 2008.

[Srivastava et al., 2011a] Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. A new representation and associated algorithms for generalized planning. Artif. Intell., 175(2):615–647, 2011.

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