Automatic Verification for Soundness of Bounded QNP Abstractions for Generalized Planning

Zhenhe Cui[1,2] , **Weidu Kuang**[1] , **Yongmei Liu**[1*]
[1]Dept. of Computer Science, Sun Yat-sen University, Guangzhou 510006, China
[2]School of CSE, Hunan University of Science and Technology, Xiangtan 411201, China
{cuizhh3, kuangwd}@mail2.sysu.edu.cn, ymliu@mail.sysu.edu.cn

Abstract

Generalized planning (GP) 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 GP. Abstractions are widely used to solve GP problems. For example, a popular abstraction model for GP is qualitative numeric planning (QNP), which extends classical planning with non-negative real variables that can be increased or decreased by some arbitrary amount. The refinement of correct solutions of sound abstractions are solutions with correctness guarantees for GP problems. More recent literature proposed a uniform abstraction framework for GP and gave model-theoretic definitions of sound and complete abstractions for GP problems. In this paper, based on the previous work, we explore automatic verification of sound abstractions for GP. Firstly, we present a proof-theoretic characterization for sound abstractions. Secondly, based on the characterization, we give a sufficient condition for sound abstractions with deterministic actions. Then we study how to verify the sufficient condition when the abstraction models are bounded QNPs where integer variables can be incremented or decremented by one. To this end, we develop methods to handle counting and transitive closure, which are often used to define numerical variables. Finally, we implement a sound bounded QNP abstraction verification system and report experimental results on several domains.

1 Introduction

Generalized planning (GP) studies the computation of general solutions for a set of planning problems [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" will eventually meet 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 GP. However, so far only limited correctness results have been obtained, mainly for the so-called $1d$ planning problems [Hu and Levesque, 2010] and extended-LL domains [Srivastava *et al.*, 2011a].

Abstractions are widely used to solve GP 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 GP 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 GP problems has attracted the attention of researchers in recent years. Bonet *et al.* [2019a] proposed learning a QNP abstraction for a GP 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 GP 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 (HL) action theory to a low-level (LL) 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 GP. They formalized a GP 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 GP problem, and showed that solutions to a GP problem are nicely related

[*]Corresponding Author

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 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 GP. First of all, we give a proof-theoretic characterization of sound abstractions for GP in the situation calculus. Secondly, based on the characterization, we give a sufficient condition for sound abstractions. Then we study how to verify the sufficient condition when the abstraction models are bounded QNPs where integer variables can be incremented or decremented by one. To address this, 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 sound bounded QNP 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

In this section, we introduce the situation calculus and Golog, regression and its two extensions, and a generalized planning abstraction framework.

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 $s \sqsubseteq s'$, 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. A situation s is executable if it is possible to perform the actions in s one by one: $Exec(s) \doteq \forall a, s'.do(a, s') \sqsubseteq s \supset Poss(a, s')$.

In the situation calculus, a particular domain of application can be specified by a basic action theory (BAT) of the following form: $\mathcal{D} = \Sigma \cup \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$, where Σ is the set of the foundational axioms for situations; $\mathcal{D}_{ap}$ 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; $\mathcal{D}_{ss}$ 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; $\mathcal{D}_{una}$ is the set of unique name axioms for actions; $\mathcal{D}_{S_0}$ 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:

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

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 [De Giacomo *et al.*, 2000]. In transition semantics, a configuration 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. 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: $Do(\delta, s, s') \doteq \exists \delta'.Trans^*(\delta, s, \delta', s') \wedge Final(\delta', s')$, 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 $\mathcal{D}$ and a formula ϕ. We use $\mathcal{R}_{\mathcal{D}}[\phi]$ to denote the formula obtained from ϕ by the following steps: for each term $f(\vec{t}, do(\alpha, \sigma))$, replace ϕ with $\exists y.\psi_f(\vec{t}, y, \alpha, \sigma) \wedge \phi[f(\vec{t}, do(\alpha, \sigma))/y]$[1]; replace each atom $P(\vec{t}, do(\alpha, \sigma))$ with $\phi_P(\vec{t}, \alpha, \sigma)$; replace each precondition atom $Poss(A(\vec{t}), \sigma)$ with $\Pi_A(\vec{t}, \sigma)$; and further simplify the result with $\mathcal{D}_{una}$.

Proposition 1. $\mathcal{D} \models \phi \equiv \mathcal{R}_{\mathcal{D}}[\phi]$.

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

Definition 2. Given a situation-suppressed formula ϕ and a Golog program δ, the extended regression $\mathcal{R}^E[\phi(s), \delta]$ can be inductively defined as follows:

- $\mathcal{R}^E[\phi(s), \alpha] = \mathcal{R}_{\mathcal{D}}[Poss(\alpha, s) \wedge \phi(do(\alpha, s))]$,
- $\mathcal{R}^E[\phi(s), \psi?] = \psi[s] \wedge \phi(s)$,
- $\mathcal{R}^E[\phi(s), \delta_1; \delta_2] = \mathcal{R}^E[\mathcal{R}^E[\phi(s), \delta_2], \delta_1]$,
- $\mathcal{R}^E[\phi(s), \delta_1|\delta_2] = \mathcal{R}^E[\phi(s), \delta_1] \vee \mathcal{R}^E[\phi(s), \delta_2]$,

[1] $\phi[t'/t]$ denotes that formula obtained from ϕ by replacing all occurrences of t' in ϕ by t.

- $\mathcal{R}^E[\phi(s), (\pi x)\delta(x)] = (\exists x)\mathcal{R}^E[\phi(s), \delta(x)]$.

Proposition 2. *Given a basic action theory $\mathcal{D}$, a Golog program δ and a situation-suppressed formula ϕ, we have:*

$$\mathcal{D} \models \mathcal{R}^E[\phi(s), \delta] \equiv \exists s'.Do(\delta, s, s') \wedge \phi[s'].$$

Li and Liu [2015] presented the universally extended regression, notation $\mathcal{R}^U[\phi(s), \delta]$ denotes a state formula expressing that all executions of program δ starting from s making ϕ hold. To get the definition of universally extended regression, one can replace the symbols $\mathcal{R}^E$, $\wedge$, $\vee$, and $\exists$ in Definition 2 with $\mathcal{R}^U$, $\supset$, $\wedge$, and $\forall$, respectively.

Proposition 3. *Given a basic action theory $\mathcal{D}$, a Golog program δ and a situation-suppressed formula ϕ, we have:*

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

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 extended the situation calculus with counting, following the logic FOCN [Kuske and Schweikardt, 2017], they introduce counting terms of the form $\#\bar{y}.\varphi$, meaning the number of tuples $\bar{y}$ satisfying formula φ; transitive closure is often used to define counting terms, following transitive closure logic [Immerman and Vardi, 1997], they introduced 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.

Now we introduce the GP abstraction framework proposed in [Cui *et al.*, 2021].

Definition 3. A GP problem is a triple $\mathcal{G} = \langle \mathcal{D}, C, G\rangle$, 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 is a goal condition.

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 fluents: $clear(x)$, $ontable(x)$, $on(x, y)$, and $holding(x)$. In this problem, the trajectory constraint C is $\top$, the goal state G is $clear(A)$, some example axioms from $\mathcal{D}$ are as follows:

Precondition Axioms.

$$Poss(mt(x), s) \equiv clear(x, s) \wedge \neg ontable(x, s);$$

Successor State Axioms.

$$on(x, y, do(a, s)) \equiv on(x, y, s) \wedge \neg a = unstack(x, y);$$

Initial Situation Axiom.

$$\exists x.on^+(x, A) \wedge ontable(A) \wedge \neg holding(x),$$

where the formula $on^+(x, A)$ is a transitive closure formula, which means that the block x is above block A.

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

Definition 4. 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 HL deterministic or non-deterministic action type A, $m(A(\vec{x})) = \delta_A(\vec{x})$, where $\delta_A(\vec{x})$ is a LL program; for each HL relational fluent P, $m(P(\vec{x})) = \phi_P(\vec{x})$, where $\phi_P(\vec{x})$ is a LL situation-suppressed formula; for each HL functional fluent f, $m(f(\vec{x})) = \tau_f(\vec{x})$, where $\tau_f(\vec{x})$ 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 model M_h is m-isomorphic to a situation s_l in a model M_l, written $s_h \sim_m s_l$, if: for any HL relational fluent P, and variable assignment v, we have $M_h, v[s/s_h] \models P(\vec{x}, s)$ iff $M_l, v[s/s_l] \models m(P)(\vec{x}, s)$; for any HL functional fluent f, variable assignment v, we have $M_h, v[s/s_h] \models f(\vec{x}, s) = y$ iff $M_l, v[s/s_l] \models m(f)(\vec{x}, s) = y$.

To relate HL and LL models, they defined two relations: m-simulation and m-back-simulation. Intuitively, 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, Δ_S^M denotes the situation domain of M; S_0^M stands for the initial situation of M; the notation $Term(\delta, s, C)$ means starting in situation s, program δ terminates under constraint C:

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

Definition 6. 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: $s_h \sim_m s_l$; for any HL action type A, and variable assignment v, $M_l, v[s/s_l] \models$ 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 HL 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.

Based on the notions above, they defined sound/complete abstraction on model and theory level. Sound abstraction means that HL behavior entails LL behavior, and complete abstraction means 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, sound and complete abstractions are defined as follows:

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

- there is a model M_h of $\mathcal{D}_h$ such that: (1) M_h is a sound m-abstraction of M_l via B_1; (2) for any situations s_h in M_h, s_l in M_l, if $\langle s_h, s_l\rangle \in B_1$ and $M_h, v[s/s_h] \models G_h[s]$, then $M_l, v[s/s_l] \models G_l[s]$, and,
- there is a model M'_h of $\mathcal{D}_h$ such that: (1) M'_h is a complete m-abstraction of M_l via B_2; (2) for any situations s_h in M'_h, s_l in M_l, if $\langle s_h, s_l\rangle \in B_2$ and $M'_h, v[s/s_h] \models G_h[s]$, then $M_l, v[s/s_l] \models G_l[s]$.

Definition 10. $\mathcal{G}_h$ is a complete m-abstraction of $\mathcal{G}_l$, if for any model M_h of $\mathcal{D}_h$:

- there is a model M_l of $\mathcal{D}_l$ such that: (1) M_h is a complete m-abstraction of M_l via B_1; (2) for any situations s_h in M_h, s_l in M_l, if $\langle s_h, s_l\rangle \in B_1$ and $M_l, v[s/s_l] \models G_l[s]$, then $M_h, v[s/s_h] \models G_h[s]$, and,
- there is a model M'_l of $\mathcal{D}_l$ such that: (1) M_h is a sound m-abstraction of M'_l via B_2; (2) for any situations s_h in M_h, s'_l in M'_l, if $\langle s_h, s'_l\rangle \in B_2$ and $M'_l, v[s/s'_l] \models G_l[s]$, then $M_h, v[s/s_h] \models G_h[s]$.

3 Proof-Theoretic Characterization

In this section, we give proof-theoretic characterizations of abstractions for GP.

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

$$anyhlas \doteq (|_{A\in\mathcal{A}_h}\ \pi\vec{x}.A(\vec{x}))^*,\ anyllps \doteq m(anyhlas).$$

We call a situation s s.t. $Do(anyllps, S_0, s)$ holds an executable refinement of a HL situation, ER in short. The following notation *Infexe*(δ, h, C) means h is an infinite execution of program δ satisfying trajectory constraint C:

$$Infexe(\delta, h, C) \doteq C(h) \wedge \forall s' \sqsubset h \exists\delta'.\ Trans^*(\delta, S_0, \delta', s').$$

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:

$$\begin{aligned} &R(s, s') \doteq \forall P.P(S_0, S_0)\wedge\\ &\quad \textstyle\bigwedge_{A\in\mathcal{A}_h}\forall\vec{x}, s, s_1, s', s'_1.(P(s, s') \wedge Do(m(A(\vec{x})), s, s_1)\\ &\qquad\quad \wedge Do(m(A(\vec{x})), s', s'_1) \supset P(s_1, s'_1)) \supset P(s, s'). \end{aligned}$$

Let ϕ be a HL formula uniform in a situation. We use $m(\phi)$ to denote the formula resulting from replacing each high-level symbol in ϕ 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 $s' = do(A(\vec{t}), s)$, where s and s' are variables.

Proposition 4. *Any trajectory constraint can be converted to an equivalent one in normal form.*

Proof. First, note that there are a finite number of action functions $A_1, \ldots, A_n$, and hence we have $\forall a \bigvee_{i=1}^n \exists\vec{x}.a = A_i(\vec{x})$. Thus, quantification over action variables can be removed as follows: replace $\exists a\phi(a)$ with $\bigvee_{i=1}^n \exists\vec{x}\phi(A_i(\vec{x}))$. Then $Poss(A_i(\vec{t}), \sigma)$ can be replaced with the instantiated RHS of the action precondition axiom for A_i. Let C be the resulting trajectory constraint. Let $do(A(\vec{t}), \sigma)$ be an appearance of do not in the required form. We replace C by $\exists s.s = do(A(\vec{t}), \sigma) \wedge C[do(A(\vec{t}), \sigma)/s]$, where $C[t_1/t_2]$ denotes the formula obtained from C by replacing any appearance of t_1 by t_2. We repeat this process until any appearance of do is in the required form. $\square$

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 $\exists s \sqsubset h$ with $\exists s \sqsubset h.Do(anyllps, S_0, s)$, then replace any appearance of $s' = do(A(\vec{t}), s)$ with $Do(m(A(\vec{t})), s, s')$, 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_l 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 infinite histories of M_h and M_l, respectively. We write $\langle h_h, h_l\rangle \in B$ if the following hold: for any $s_h \sqsubset h_h$, there is an ER $s_l \sqsubset h_l$ s.t. $\langle s_h, s_l\rangle \in B$; and for any ER $s_l \sqsubset h_l$, there is $s_h \sqsubset h_h$ s.t. $\langle s_h, s_l\rangle \in B$.

Proposition 5. *Let M_h be a sound or complete abstraction of M_l via B. Let $\langle h_h, h_l\rangle \in B$. Let C be a HL trajectory constraint. Then $M_h, v[h/h_h] \models C(h)$ iff $M_l, v[h/h_l] \models m(C)(h)$.*

Proof. WLOG, we assume that C is in normal form. We prove by induction that for any sub-formula C' of C, for any free variables $\vec{s}$ and $\vec{x}$, and for any sub-situations $\vec{s}_h$ of h_h and any ER sub-situations $\vec{s}_l$ of h_l s.t. $\langle s^i_h, s^i_l\rangle \in B$ for any index i, we have $M_h, v[h/h_h, \vec{s}/\vec{s}_h] \models C'$ iff $M_l, v[h/h_l, \vec{s}/\vec{s}_l] \models m(C')$. The base cases are of the following forms $s \sqsubseteq s'$, $s' = do(A(\vec{t}), s)$, $P(\vec{x}, s)$, and $f(\vec{x}, s) = y$. All these cases can be easily proved. The induction cases are those of $\neg$, $\wedge$, $\exists x$, and $\exists s \sqsubset h$. We only prove the last case, the other cases are easy. So suppose $M_h, v[h/h_h, \vec{s}/\vec{s}_h] \models \exists s \sqsubset h.C'$. Then there is $s_h \sqsubset h_h$ s.t. $M_h, v[h/h_h, \vec{s}/\vec{s}_h, s/s_h] \models C'$. By Definition 13, there is an ER $s_l \sqsubset h_l$ s.t. $\langle s_h, s_l\rangle \in B$. By induction hypothesis, $M_h, v[h/h_h, \vec{s}/\vec{s}_h, s/s_h] \models C'$ iff $M_l, v[h/h_l, \vec{s}/\vec{s}_l, s/s_l] \models m(C')$. Thus $M_l, v[h/h_l, \vec{s}/\vec{s}_l] \models \exists s \sqsubset h.Do(anyllps, S_0, s) \wedge m(C')$, which is $m(\exists s \sqsubset h.C')$. The other direction can be similarly proved. $\square$

Non-deterministic actions in [Cui *et al.*, 2021] are treated as non-deterministic Golog programs. In particular, each non-deterministic action A has a definition in the form $A(\vec{x}) \doteq \pi\vec{u}.A_d(\vec{x}, \vec{u})$, where A_d is a deterministic action. We let $\Pi_A(\vec{x}, s)$ denote $\exists\vec{u}.\Pi_{A_d}(\vec{x}, \vec{u}, s)$, let $\phi_{P,A_d}(\vec{y}, \vec{x}, \vec{u}, s)$ denote $\phi_P(\vec{y}, A_d(\vec{x}, \vec{u}), s)$ simplified by using $\mathcal{D}_{una}$, and let $\psi_{f,A_d}(\vec{y}, z, \vec{x}, \vec{u}, s))$ denote $\psi_f(\vec{y}, z, A_d(\vec{x}, \vec{u}), s)$ simplified by using $\mathcal{D}_{una}$.

We now introduce the following abbreviations:

$$\psi_T \doteq \textstyle\bigwedge_{A\in\mathcal{A}_h} \forall\vec{x}.Term(m(A(\vec{x})), s, C_l),$$

$$\xi_P \doteq \textstyle\bigwedge_{A\in\mathcal{A}_h} \forall\vec{x}, s'.Do(m(A(\vec{x})), s, s') \supset \exists\vec{u}.$$

$\bigwedge_{P\in\mathcal{P}_h}[\forall\vec{y}.m(P(\vec{y},s'))\equiv m(\phi_{P,A_d}(\vec{y},\vec{x},\vec{u},s))]$,
$\xi_f \doteq \bigwedge_{A\in\mathcal{A}_h}\forall\vec{x},s'.Do(m(A(\vec{x})),s,s')\supset\exists\vec{u}.$
$\bigwedge_{f\in\mathcal{F}_h}[\forall\vec{y},z.m(f(\vec{y},s')=z)\equiv m(\psi_{f,A_d}(\vec{y},z,\vec{x},\vec{u},s))]$,
where, ψ_T says that the refinement of any HL action terminates in s under C_l; ξ_P and ξ_f says that for any HL action $A(\vec{x})$, if its refinement transforms situation s to s', then there is $\vec{u}$ s.t. the mapping of all SSAs instantiated with $A_d(\vec{x},\vec{u})$ 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 $\mathcal{D}_l$ entails that for any ER, *i.e.*, executable refinement of a HL situation, the executability of the refinement of any HL action implies its mapped precondition; Item 3 says that $\mathcal{D}_l$ entails that for any ER, 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 $\mathcal{D}_l$ entails that the existence of an infinite execution of $anyllps$ satisfing the LL constraint is equivalent to the existence of one satisfying the mapped HL constraint.

Theorem 1. *$\mathcal{G}_h$ is a sound m-abstraction of $\mathcal{G}_l$ iff the following conditions hold:*

1. $\mathcal{D}_l^{S_0}\models m(\phi)$, *where* $\phi\in\mathcal{D}_h^{S_0}$;
2. $\mathcal{D}_l\models\forall s.Do(anyllps,S_0,s)\supset$ $\bigwedge_{A\in\mathcal{A}_h}\forall\vec{x},s'.Do(m(A(\vec{x})),s,s')\supset m(\Pi_A(\vec{x},s))$;
3. $\mathcal{D}_l\models\forall s.Do(anyllps,S_0,s)\supset$ $\bigwedge_{A\in\mathcal{A}_h}\forall\vec{x}.m(\Pi_A(\vec{x},s))\supset$ $\exists s',s''.R(s,s')\wedge Do(m(A(\vec{x})),s',s'')$;
4. $\mathcal{D}_l\models\forall s.Do(anyllps,S_0,s)\supset\psi_T\wedge\xi_P\wedge\xi_f$;
5. $\mathcal{D}_l\models\exists h_l.Infexe(anyllps,h_l,C_l)\equiv$ $\exists h_l.Infexe(anyllps,h_l,m(C_h))$;
6. $\mathcal{D}_l\models\forall s.Do(anyllps,S_0,s)\wedge m(G_h)[s]\supset G_l[s]$.

Proof. $(sketch)$ $\Rightarrow$: Item 1 and 6 are easy to prove. For Item 2 and 4, according to Definition 9, for each model M_l of $\mathcal{G}_l$, there exist a model M_h of $\mathcal{G}_h$ s.t. M_h is a complete abstraction of M_l via a m-simulation relation B_1. Let s_l be a LL situation which satisfies $Do(anyllps,S_0,s)$, then based on Definition 6, there is a HL situation s_h s.t. $\langle s_h,s_l\rangle\in B_1$, and for any HL action type A: if $m(A(\vec{x}))$ is executable in s_l, then $A(\vec{x})$ is executable in s_h, thus, s_l satisfies $m(\Pi_A(\vec{x},s))$; s_l satisfies $Term(m(A(\vec{x})),s,C_l)$; if there exists a LL situation s_l' satisfies $Do(m(A(\vec{x})),s_l,s_l')$, then there exists an action $A(\vec{x},\vec{u})$ and a HL situation s_h', s.t. $\langle s_h',s_l'\rangle\in B_1$, furthermore, if s_h and s_h' satisfy the HL fluent-action related SSAs, then s_l and s_l' satisfy the mapped HL fluent-action related SSAs. For Item 3, we can prove that M_h is also a sound m-abstraction of M_l via a m-back-simulation relation B_2, then for $\langle s_h,s_l\rangle\in B_1$, there also exists a LL situation s_l', such that $\langle s_h,s_l'\rangle\in B_2$, thus, s_l and s_l' are R-related, furthermore, $A(\vec{x})$ is executable in s_h, then $m(A(\vec{x}))$ is executable in s_l'. Finally, let h_l be a LL infinite history, then there exists a HL infinite history h_h, based on the relation B_2, we can construct another LL infinite history h_l', this coupled with Proposition 5 can imply Item 5.

$\Leftarrow$: For any LL model M_l, by using refinement mapping, we first construct a HL model M_h, then based on the Item 1, 2, 4, and 5, we can construct a m-simulation relation B_1 between M_h and M_l. Similarly, based on the Item 1, 3, 4, and 5, we can construct a m-back-simulation relation B_2 between M_h and M_l. Then, we can get that $\mathcal{G}_h$ is sound abstraction of $\mathcal{G}_l$ based on Item 6 and Definition 9. □

Complete abstraction on the theory level means that for each HL model M_h, there exist two LL models M_l, M_l' s.t. M_h is a sound abstraction of M_l, and a complete abstraction of M_l'. However, in this case, M_l and M_l' may be different, and thus we cannot give a proof-theoretic characterization for complete abstractions. Nonetheless, we give a sufficient and necessary condition for complete abstractions in terms of the existence of two LL models M_l and M_l' satisfying 5 conditions where Item 2 says that M_l satisfies for any ER, ψ_T, ξ_P, and ξ_f hold, and the LL goal implies the mapped HL goal, and the executability of the refinement of any HL action implies its mapped precondition; Item 3 says that M_l' satisfies that there exists a set P of situations including the initial situation such that for any P-situation, ψ_T, ξ_P, and ξ_f hold, and the LL goal implies the mapped HL goal, and the mapped precondition of any HL action implies that its refinement is executable and leads to a P-situation.

Theorem 2. *$\mathcal{G}_h$ is a complete m-abstraction of $\mathcal{G}_l$ iff for any model M_h of $\mathcal{D}_h$, there are two models M_l, M_l' of $\mathcal{D}_l$ s.t.*

1. $S_0^{M_h}\sim_m S_0^{M_l}$, $S_0^{M_h}\sim_m S_0^{M_l'}$;
2. $M_l\models\forall s.Do(anyllps,S_0,s)\supset$ $\bigwedge_{A\in\mathcal{A}_h}[\forall\vec{x},s'.Do(m(A(\vec{x})),s,s')\supset m(\Pi_A(\vec{x},s))]$ $\wedge\psi_T\wedge\xi_P\wedge\xi_f\wedge[G_l[s]\supset m(G_h)[s]]$;
3. $M_l'\models\exists P.P(S_0)\wedge\forall s.P(s)\supset$ $\bigwedge_{A\in\mathcal{A}_h}\forall\vec{x}.m(\Pi_A(\vec{x},s))\supset\exists s'.Do(m(A(\vec{x})),s,s')$ $\wedge P(s')\wedge\psi_T\wedge\xi_P\wedge\xi_f\wedge[G_l[s]\supset m(G_h)[s]]$;
4. *if* $M_l\models\exists h_l.Infexe(anyllps,h_l,C_l)$, *then* $M_h\models\exists h_h.Infexe(anyhlas,h_h,C_h)$;
5. *if* $M_h\models\exists h_h.Infexe(anyhlas,h_h,C_h)$, *then* $M_l'\models\exists h_l.Infexe(anyllps,h_l,C_l)$.

Proof. $(sketch)$ $\Rightarrow$: Item 1 is easy to prove. For Item 2, according to the Definition 10, we know that for each HL model M_h of $\mathcal{D}_h$, there exist a LL model M_l, such that M_h is a complete abstraction of M_l via a m-simulation relation B_1. Let s_l be a LL situation which satisfies $Do(anyllps,S_0,s)$, then based on Definition 6, there is a HL situation s_h s.t. $\langle s_h,s_l\rangle\in B_1$, and for any HL action type A: if $m(A(\vec{x}))$ is executable in s_l, then $A(\vec{x})$ is executable in s_h, thus, s_l satisfies $m(\Pi_A(\vec{x},s))$; s_l satisfies $Term(m(A(\vec{x})),s,C_l)$; if there exists a LL situation s_l' satisfies $Do(m(A(\vec{x})),s_l,s_l')$, then there exists an action $A(\vec{x},\vec{u})$ and a HL situation s_h', s.t. $\langle s_h',s_l'\rangle\in B_1$, furthermore, if s_h and s_h' satisfy the HL fluent-action related SSAs, then s_l and s_l' satisfy the mapped HL fluent-action related SSAs; if s_l satisfies G_l, then s_h satisfies G_h, and hence, s_l satisfies $m(G_h)$. For Item 3, according to the Definition 10, we know that for the HL model M_h, there is another LL model M_l', such that M_h is a sound abstraction of M_l' via a m-back-simulation relation B_2. We construct the situation set P of M_l' based on the following two steps: (i) $S_0^{M_l'} \in P$; (ii) for any LL program $anyllps$, if there is a situation s_l satisfies $Do(anyllps, S_0^{M_l'}, s_l)$, then $s_l \in P$. Now we let s_h be a reachable situation of M_h via executing $anyhlas$, then according to the construction of P, there is a situation $s_l \in P$ of M_l', such that $\langle s_h, s_l\rangle \in B_2$, and for any HL action type A: if $A(\vec{x})$ is executable in s_h, then $m(A(\vec{x}))$ is executable in s_l, thus, s_l satisfies $m(\Pi_A(\vec{x}, s))$; s_l satisfies $Term(m(A(\vec{x})), s, C_l)$; if there exists a HL situation s_h' satisfies $Do(A(\vec{x}), s_h, s_h')$, then there exists an action $A(\vec{x}, \vec{u})$ and a LL situation $s_l' \in P$, s.t. $\langle s_h', s_l'\rangle \in B_2$, furthermore, if s_h and s_h' satisfy the HL fluent-action related SSAs, then s_l and s_l' satisfy the mapped HL fluent-action related SSAs; if s_l satisfies G_l, then s_h satisfies G_h, and hence, s_l satisfies $m(G_h)$. Item 4 and 5 can be proved by using the Definition 8 and 7, respectively.

$\Leftarrow$: For the model M_h and M_l, based on the Item 1, 2, and 4, we can construct a m-simulation relation B_1 between M_h and M_l. Similarly, based on the Item 1, 3, and 5, we can construct a m-back-simulation relation B_2 between M_h and M_l. Then, we can get that $\mathcal{G}_h$ is complete abstraction of $\mathcal{G}_l$ based on the Definition 10. $\square$

4 Automatic Verification for Soundness of Bounded QNP Abstractions

In this section, we explore how to automatically verify the soundness of bounded QNP abstractions with theorem provers. To this end, we first introduce some restrictions in Theorem 1, and give a sufficient condition for sound abstractions, which is verifiable in first-order logic with counting and transitive closure. Then, based on the sufficient condition, we present the methodology of sound bounded QNP abstraction verification with theorem provers. Particularly, we develop methods to handle counting and transitive closure.

4.1 A Sufficient Condition for Sound Abstraction

We first discuss how to obtain a sufficient condition for sound abstractions from Theorem 1:

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

(2). Determining whether a given program terminates is the quintessential undecidable problem. We assume that any HL action refinement does not involve iterations. Then, the formula ψ_T in task 4 is true trivially;

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

(4). Since tasks 2 and 3 involve the reasoning about LL Golog programs, we use existentially extended regression to compute executability conditions of Golog programs. Given a program δ and a situation s, the executability condition $pre(\delta, s)$ of δ in the situation s can be computed as $\mathcal{R}^E[\top(s), A(\vec{x})]$. Thus, in tasks 2 and 3, we can equivalently replace $Do(m(A(\vec{x}), s, s'))$ by $pre(m(A(\vec{x})), s)$. In addition, we introduce the following abbreviation:

$$\xi_A \doteq \textstyle\bigwedge_{A\in\mathcal{A}_h} \forall\vec{x}.pre(m(A(\vec{x})), s) \equiv m(\Pi_A(\vec{x}, s)),$$

which means that for any HL action $A(\vec{x})$ and a LL situation s, $m(A(\vec{x}))$ is executable in s if and only if s satisfies the refinement of precondition of the action $A(\vec{x})$. Then, for getting a sufficient condition for sound abstractions, we verify a stronger task which can implies both task 2 and 3: $\models_{foct} \forall s.\mathcal{D}_{sc}(s) \supset \xi_A$, where, $\models_{foct}$ denotes the entailment in first-order logic with counting and transitive closure;

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

$$\begin{aligned}\zeta_P \doteq\ & \textstyle\bigwedge_{A\in\mathcal{A}_h} \forall\vec{x}.pre(m(A(\vec{x}), s))\\ &\supset \textstyle\bigwedge_{P\in\mathcal{P}_h} \forall\vec{y}.[\mathcal{R}^E[m(P(\vec{y}))[s], m(A(\vec{x}))]\\ &\supset m(\phi_{P,A}(\vec{x}, \vec{y}, s))] \wedge [m(\phi_{P,A}(\vec{x}, \vec{y}, s))\\ &\supset \mathcal{R}^U[m(P(\vec{y}))[s], m(A(\vec{x}))],\end{aligned}$$

$$\begin{aligned}\zeta_f \doteq\ & \textstyle\bigwedge_{A\in\mathcal{A}_h} \forall\vec{x}.pre(m(A(\vec{x}), s))\\ &\supset \textstyle\bigwedge_{f\in\mathcal{F}_h} \forall\vec{y}, z.[\mathcal{R}^E[m(f(\vec{y}) = z)[s], m(A(\vec{x}))]\\ &\supset m(\psi_{f,A}(\vec{x}, \vec{y}, z, s))] \wedge [m(\psi_{f,A}(\vec{x}, \vec{y}, z, s))\\ &\supset \mathcal{R}^U[m(f(\vec{y}) = z)[s], m(A(\vec{x}))]],\end{aligned}$$

where, ζ_P says that if the LL program $m(A(\vec{x}))$ is executable in situation s, then there exists an execution of $m(A(\vec{x}))$ starting from s making $m(P(\vec{y}))$ hold, and all executions of $m(A(\vec{x}))$ starting from s making $m(P(\vec{y}))$ hold. ζ_f and ζ_P have similar meaning.

Based on the two extended regression definitions, we can get that task 4 is equivalent to $\models_{foct} \forall s.\mathcal{D}_{sc}(s) \supset \zeta_P \wedge \zeta_f$.

Finally, we can get the following corollary.

Corollary 1. *Given a GP problem $\mathcal{G}_l$ and its abstraction $\mathcal{G}_h$, suppose that all the HL actions are deterministic and their refinements do not involve iteration, then $\mathcal{G}_h$ is a sound abstraction of $\mathcal{G}_l$ if:*

1. $\mathcal{D}_l^{S_0} \models_{foct} m(\phi)$, *where* $\phi \in \mathcal{D}_h^{S_0}$*;*
2. $\models_{foct} \forall s.\mathcal{D}_{sc}(s) \supset \xi_A$*;*
3. $\models_{foct} \forall s.\mathcal{D}_{sc}(s) \supset \zeta_P \wedge \zeta_f$*;*
4. $\models_{foct} \forall s.\mathcal{D}_{sc}(s) \wedge m(G_h)[s] \supset G_l[s]$.

Note that when conditions of Corollary 1 do not hold, we cannot determine whether an abstraction is sound.

4.2 Sound Bounded QNP Abstraction Verification

QNPs are classical planning problems extended with non-negative real variables and non-deterministic actions that increase or decrease the values of variables by positive indeterminate amounts. QNPs were first introduced by Srivastava *et al.* [2011b], and nowadays, they have been widely used for modeling abstractions of GP problems.

Given a set of propositional variables F and a set of non-negative numerical variables V, we refer to p and $\neg p$ for $p \in F$ as the F-literals, $v > 0$ and $v = 0$ for $v \in V$ as the V-literals. A pair of complementary literals has the form $\{p, \neg p\}$ for $p \in F$, or $\{v > 0, v = 0\}$ for $v \in V$. A set of literals is consistent if it does not contain complementary literals. Let $\mathcal{L}_X$ (resp. $\mathcal{L}_F$) denote the set of all consistent sets of literals from $F \cup V$ (resp. F).

Definition 14. A QNP problem $Q = \langle F, V, I, O, G \rangle$ consists of F, a set of propositional variables; V, a set of non-negative real variables; $I \in \mathcal{L}_X$, a set of initial true literals; $G \in \mathcal{L}_X$, the set of goal literals; and O, a set of actions. Every $a \in O$ has a set of preconditions $pre(a) \in \mathcal{L}_X$, a set of propositional effects $\mathit{eff}(a) \in \mathcal{L}_F$, and a set of numerical effects $N(a)$ which contain special atoms of the form $inc(v)$ or $dec(v)$ to increase or decrease v by some arbitrary amount for $v \in V$. Actions with the $dec(v)$ effect must feature the precondition $v > 0$ for any variable $v \in V$.

Many abstraction models of GP problems involve integer variables. For example, Bonet and Geffner [2018] proposed abstracting generalized classical planning problems into a variant of QNPs where the numerical variables of QNPs are all non-negative integer variables. In this work, we consider that abstractions of GP problems are bounded QNPs where integer variables can be incremented or decremented by one.

Definition 15. A bounded QNP problem is a 5-tuple $Q = \langle F, V, I, O, G \rangle$ where F is a set of propositional variables; V is a set of non-negative integer variables; $I \in \mathcal{L}_X$ is a set of initial true literals; $G \in \mathcal{L}_X$ is the set of goal literals; and O is a set of actions. Every $a \in O$ has a set of preconditions $pre(a) \in \mathcal{L}_X$, a set of propositional effects $\mathit{eff}(a) \in \mathcal{L}_F$ and a set of numerical effects $N(a)$ of the form $v \uparrow$ ($v \downarrow$) for $v \in V$, which means that v is incremented (decremented) by 1. Actions with the $v \downarrow$ effect must feature the precondition $v > 0$ for any variable $v \in V$.

Example 1 cont'd. An abstraction for the GP problem $ClearA$ is a bounded QNP $Q = \langle F, V, I, O, G \rangle$, 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 $n > 0 \wedge \neg H$; the goal state G is $n = 0$; the actions in $O = \{pickabove, putaside\}$ are defined as follows:

$$pickabove : \langle \neg H \wedge n > 0; H, n-1 \rangle;\ putaside : \langle H; \neg H \rangle.$$

In the bounded QNP abstraction case, the verification for tasks 1 and 2 can be easily transformed into two first-order theorem proving tasks with transitive closure; the verification for task 3 in Corollary 1 can be more specific, concretely, it can be divided as the following two tasks:

For each HL relational fluent $p \in \mathcal{P}_h$, and HL action $A \in \mathcal{A}_h$, A makes p true (the false case can be discussed similarly) means that in LL, all the executions of $m(A)$ make $m(p)$ true, then we have the following task 3^*:

$$\models_{fot} \forall s.\mathcal{D}_{sc}(s) \wedge \\ \quad \forall \vec{x}.pre(m(A(\vec{x})), s) \supset \mathcal{R}^U[m(p(s)), m(A(\vec{x}))],$$

where, the notation $\models_{fot}$ means the first-order entailment with transitive closure.

For each HL functional fluent $n \in \mathcal{F}_h$, and HL action $A \in \mathcal{A}_h$, if A makes n increase by 1 (the decrease case can be discussed similarly), then for any LL situation s and any situation s' that is arrived by executing $m(A(\vec{x}))$ from s, we have $m(n)[s'] = m(n)[s] + 1$. Assuming that $m(n) = \#x.\phi(x)$, we have the following formula Ψ holds:

$$[\exists x.\phi(x, s') \wedge \neg\phi(x, s)] \wedge [\forall x.\phi(x, s) \supset \phi(x, s')] \wedge \\ [\forall x, y.\phi(x, s) \wedge \neg\phi(x, s') \wedge \phi(y, s') \wedge \phi(y, s) \supset x = y],$$

which means that there exists only one object that makes $\phi(x)$ true from false after the execution of the program $m(A(\vec{x}))$. Then based on Proposition 3, we have the following task $3^\#$:

$$\models_{fot} \forall s.\mathcal{D}_{sc}(s) \wedge \\ \quad \forall \vec{x}.pre(m(A(\vec{x})), s) \supset \mathcal{R}^U[\Psi(s), m(A(\vec{x}))].$$

Transitive closure formulas are often used to define counting terms. Thus, task $3^\#$ 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 ϕ: $[TC_{x,y}on(x, y)](x, C, s')$, the regression result $\mathcal{R}_\mathcal{D}[\phi]$ of ϕ related to the concrete action $unstack(A, B)$ is as follows:

$$[TC_{x,y}on(x, y) \wedge (x \neq A \vee y \neq B)](x, C, s),$$

where, $s' = do(unstack(A, B), s)$.

Existing first-order theorem provers cannot express the transitive closure of formulas with the form $[TC_{x,y}\varphi](u, v)$. To handle this case, we first define the formula $\varphi(x, y)$ as a new relation $P(x, y)$, then equivalently replace the formula $[TC_{x,y}\varphi](u, v)$ with $P^+(x, y)$, and finally feed it into a theorem prover. Furthermore, existing theorem provers cannot express the minimality of transitive closure as well, thus, we can only verify the weak transitivity.

5 Verification System and Experiment Results

In this section, we designed a sound bounded QNP abstraction verification system[2]. The inputs of our system include a GP problem coupled with state constraints, a refinement mapping, and an abstraction problem. The output of our system is $True$ / $Unknown$. GP problems in our system take the form of STRIPS-like problems. The only extension of GP problems to STRIPS problems is that their initial states can be first-order formulas with transitive closure (see Example 1). The formalization of bounded QNP abstractions in our system is similar to that in [Bonet and Geffner, 2020].

The workflow of our verification system is as follows:

Step 1. Given the input GP problem, the system automatically generates the LL BAT;

Step 2. Based on the abstraction problem, refinement mapping, LL BAT, and LL state constraints, the system generates the verification tasks that we mention in Corollary 1. Concretely, the system generates task 1 for the HL and LL initial states; task 2 for each HL action; task 3^* for each pair of HL relational fluent and action; task $3^\#$ for each pair of HL functional fluent and action; task 4 for the HL and LL goals.

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 $Unknown$.

We use the following example to demonstrate the verification process of our system:

[2]The system is available at https://github.com/sysulic/AVS.

Example 2. This problem involves a robot with two grippers whose goal is to move some balls from room R into room D. Each gripper may carry one ball at a time. The LL predicates are: $at(r)$, $in(b, r)$, $free(g)$, and $carry(g, b)$. Actions are: $move(r, r')$, $pick(b, r, g)$, and $drop(b, r, g)$. The initial state is $at(R) \wedge \forall g.free(g) \wedge \exists b.in(b, R)$.

The domain constraint $\mathcal{D}_{sc}$ of this problem is as follows:

- $\forall g \exists b.carry(b, g) \equiv \neg free(g)$,
- $at(R) \equiv \neg at(D)$, $\neg at(R) \equiv at(D)$,
- $\forall b, g.in(b, R) \equiv \neg in(b, D) \wedge \neg carry(b, g)$,
- $\forall b, g.in(b, D) \equiv \neg in(b, R) \wedge \neg carry(b, g)$,
- $\forall b, g.carry(b, g) \equiv \neg in(b, R) \wedge \neg in(b, D)$,
- $\forall b_1, b_2, g.carry(b_1, g) \wedge carry(b_2, g) \supset b_1 = b_2$.

The bounded QNP problem in HL involves three numerical variables B, C, and G that denote, respectively, the number of balls at the source room, the number of balls carried by grippers, the number of free grippers. Boolean variable T means the robot is in the target room. The initial state is $B > 0 \wedge C = 0 \wedge G > 0 \wedge \neg T$, and the goal condition is $B = 0$. Actions are as follows:

- $Move : \langle \neg T; T \rangle$, $Leave : \langle T; \neg T \rangle$,
- $Pick : \langle \neg T, B > 0, G > 0; B - 1, C + 1, G - 1 \rangle$,
- $Drop : \langle T, C > 0;\ C - 1, G + 1 \rangle$.

The refinement mapping is as follows:

- $m(B) = \#b.in(b, R)$, $m(G) = \#g.free(g)$,
- $m(C) = \#b.\exists g.carry(b, g)$, $m(T) = at(D)$,
- $m(Move) = (at(R))?; move(R, D)$,
- $m(Leave) = (at(D))?; move(D, R)$,
- $m(Drop) =$
  $\pi b, g.(carry(b, g) \wedge at(D))?; drop(b, D, g)$
- $m(Pick) =$
  $\pi b, g.(at(R) \wedge in(b, R) \wedge free(g))?; pick(b, R, g)$.

The system automatically generates 13 tasks for this problem, some of them are as follows:

- verification involves *initial* state:
  $\models_{foct} \mathcal{D}_{sc} \wedge at(R) \wedge \forall g.free(g) \wedge \exists b.in(b, R) \supset \exists b.in(b, R) \wedge \neg \exists b, g.carry(b, g) \wedge \exists g.free(g) \wedge \neg at(D)$,
- verification involves *goal* state:
  $\models_{foct} \mathcal{D}_{sc} \wedge \forall b.in(b, D) \supset \forall b.\neg in(b, R) \wedge \forall g.free(g)$,
- verification involves precondition of $Pick$:
  $\models_{foct} \mathcal{D}_{sc} \wedge \exists b, g.in(b, R) \wedge at(R) \wedge free(g)$
  $\supset \neg at(D) \wedge \exists b.in(b, R) \wedge \exists g.free(g)$,
- verification involves the fluent-action pair $(T, Move)$:
  $\models_{foct} \forall s.\mathcal{D}_{sc}(s) \wedge pre(m(Move), s)$
  $\supset \mathcal{R}^U[m(T(s)), m(Move)]$, where,
  - $pre(m(Move), s) \equiv at(R)$,
  - $\mathcal{R}^U[m(T(s)), m(Move)] \equiv at(R) \supset (at(R) \supset (\neg at(D) \wedge D = D) \vee (at(D) \wedge \neg D = R))$.

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

Table 1: Experimental Results

Our verification system was also tested on other 6 domains. Abstractions of **ClearA**, **OnAB** come from [Bonet *et al.*, 2019a]. Abstractions of other problems are provided by us.

OnAB is about achieving the goal $On(A, B)$ in blocksworld domain where a 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 source location and transport them to the target location; **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).

Our experiments were run on a Windows machine with a 3.7GHz CPU and 16GB RAM, the default time limit of each subtask was 15s. 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 is the total time costs of all verification tasks. Experimental results showed that our system was able to successfully verify soundness of abstractions for all domains in a reasonable time.

6 Conclusion

In GP, 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 GP. We gave a proof-theoretic characterization of sound abstractions for GP in the situation calculus. Then, we got a sufficient condition for sound abstractions. To implement it in the bounded QNP abstraction case,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 Bjørner, 2008] Leonardo Mendonça de Moura and Nikolaj Bjørner. 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 *Proceeding 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.