Automated Verification of Propositional Agent Abstraction for Classical Planning via CTLK Model Checking

Kailun Luo

School of Cyberspace Security, Dongguan University of Technology, Dongguan 523808, China
luokl@dgut.edu.cn

Abstract

Abstraction has long been an effective mechanism to help find a solution in classical planning. Agent abstraction, based on the situation calculus, is a promising explainable framework for agent planning, yet its automation is still far from being tackled. In this paper, we focus on a propositional version of agent abstraction designed for finite-state systems. We investigate the automated verification of the existence of propositional agent abstraction, given a finite-state system and a mapping indicating an abstraction for it. By formalizing sound, complete and deterministic properties of abstractions in a general framework, we show that the verification task can be reduced to the task of model checking against CTLK specifications. We implemented a prototype system, and validated the viability of our approach through experimentation on several domains from classical planning.

Introduction

Abstraction has long been an effective mechanism to help find a solution in classical planning. One trend is abstraction refinement [Sacerdoti 1974; Knoblock 1994; Seipp and Helmert 2018], whose idea is to first resort to an abstract problem and obtain an abstract solution, and then refine it to a concrete solution to the original problem.

Recently, an explainable abstraction framework called *agent abstraction* was proposed by Banihashemi, Giacomo, and Lespérance [2017]. It is a first-order framework based on the situation calculus action theories [McCarthy and Hayes 1969; Reiter 2001] and the ConGolog programming language [Giacomo, Lespérance, and Levesque 2000]. The key component is a refinement mapping, which associates an abstract fluent to a concrete formula, and maps an abstract action to a concrete ConGolog program. Thus one can take an abstract view of an agent's possible concrete behaviour, suppressing the details and explaining the meanings at a high level.

While agent abstraction is a promising framework for the explainability of abstractions for classical planning, its automation is still far from being tackled. Luo et al. [2020] showed that agent abstraction can be characterized theoretically via *forgetting* [Lin and Reiter 1994]. However, an automated process is unknown, even in the propositional case. A key problem towards full automation is the existence problem of agent abstraction, i.e., whether there exists an abstract action theory being the desired abstraction, when given a concrete action theory and a refinement mapping which indicates an abstraction. Take the blocks world for example, where an agent is re-assembling blocks stacked on a table. We could specify a refinement mapping, where an abstract action $move\_to\_table$ means a concrete program moving a block onto the table, and an abstract fluent $all\_ontable$ denotes a property that all the blocks are on the table. We might want to know whether this refinement mapping is *correct*, i.e., whether there exists an abstract action theory (describing the precondition of $move\_to\_table$ and its effect on $all\_ontable$) such that if performing $move\_to\_table$ make $all\_ontable$ hold, then at the concrete level, there exists a sequence of actions resulting in a scenario where all the blocks are on the table.

The existence problem of agent abstraction is natural, as specifying a refinement mapping is easier than specifying an abstract action theory, without caring about the preconditions and effects of abstract actions. In classical planning, providing refinement mappings is relevant to exploiting *domain control knowledge* [Baier, Fritz, and McIlraith 2007] and *hierarchical task networks* [Nau et al. 2003]. Agent abstraction has its peculiarity, mainly in that with a strong theoretical guarantee, a refinement mapping can be reused clearly for any other planning problems if the refinement mapping is verified to be correct for them.

In this paper, we focus on a propositional version of agent abstraction, designed for finite-state systems and suitable for domains in classical planning. We investigate the automated verification of the existence of propositional agent abstraction, given a finite-state system and a refinement mapping.

We first formalize sound, complete and deterministic properties of abstractions in a general framework for finite-state systems. It can be viewed as model-level abstractions (compared to theory-level abstractions in agent abstraction). Roughly speaking, sound abstraction says that if there is a solution at the abstract level, then this solution can be refined to a solution at the concrete level. Complete abstraction says that if there is a solution at the concrete level, and it is *abstractable* via the refinement mapping, then this solution can be abstracted to a solution at the abstract level. Deterministic abstraction means that at the

abstract level, the effects of actions are deterministic. We show that propositional agent abstraction is a strong version of sound, complete and non-deterministic abstraction.

We then show that verifying the existence of propositional agent abstraction can be reduced to the task of model checking a logic of knowledge and branching time, CTLK [Fagin et al. 1995; Lomuscio and Raimondi 2006]. The key observation for verification is that for any state needed to be abstracted, if there is a sequence of *transitions*, then any *indistinguishable* state should have an indistinguishable sequence of transitions. With a finite-state system S and a refinement mapping, we enrich S with an indistinguishable relation to perform CTLK model checking. We implemented a prototype system, making use of the state-of-the-art tool for CTLK model checking, and validated the viability of our approach through experimentation on several domains from classical planning.

Preliminaries

Labelled Transition Systems

In this paper, we focus on finite-state systems, which are represented by labelled transition systems (LTSs) as follows.

Definition 1. A labelled transition system $\mathcal{T}$ is a tuple $(Q, q_0, Act, Tr, \Delta, l)$, where

- Q is a finite set of states;
- $q_0 \in Q$ is the starting state;
- Act is a set of actions;
- $Tr \subseteq Q \times Act \times Q$ is a serial transition relation;
- Δ is a set of atomic propositions;
- $l : Q \to 2^{\Delta}$ is a labelling function.

An LTS is *deterministic* if for any q, q', q'' and a, if $(q, a, q') \in Tr$ and $(q, a, q'') \in Tr$, then $q' = q''$; otherwise, it's non-deterministic. A transition system (TS) is an LTS without actions and whose transition relation is $Tr \subseteq Q \times Q$. A *run* of a TS is an infinite sequence of states, starting from the initial state q_0 and following the transition relation Tr.

CTLK Model Checking

Our verification relies on model checking against CTLK specifications [Fagin et al. 1995; Lomuscio and Raimondi 2006]. For our purposes, we define the syntax of logic CTLK by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{AX}\varphi \mid \mathsf{AG}\varphi \mid \mathbf{K}\varphi$$

The validity judgement for CTLK formulas is written $\mathcal{T}, r, n \models \varphi$, where $\mathcal{T}$ is an *epistemic transition system* (ETS), which denotes a TS with an *indistinguishable relation* $\sim$ over states, r is a run of the ETS, and n is a natural number. We use $r(n)$ to denote the nth state of r. Given r and r', we use $r =_{[i...j]} r'$ to express that for any $k \in [i, j]$, there is $r(k) = r'(k)$. Validity is defined as follows:

$\mathcal{T}, r, n \models p$ iff $p \in l(r(n))$;

$\mathcal{T}, r, n \models \neg\varphi$ iff $\mathcal{T}, r, n \not\models \varphi$;

$\mathcal{T}, r, n \models \varphi_1 \vee \varphi_2$ iff $\mathcal{T}, r, n \models \varphi_1$ or $\mathcal{T}, r, n \models \varphi_2$;

$\mathcal{T}, r, n \models \mathsf{AX}\varphi$ iff for any r' s.t. $r' =_{[0...n]} r$,
there is $\mathcal{T}, r', n+1 \models \varphi$;

$\mathcal{T}, r, n \models \mathsf{AG}\varphi$ iff for any r' s.t. $r' =_{[0...n]} r$ and for any n' s.t. $n' \geq n$, there is $\mathcal{T}, r', n' \models \varphi$;

$\mathcal{T}, r, n \models \mathbf{K}\varphi$ iff for any r' and n' s.t. $r(n) \sim r'(n')$, there is $\mathcal{T}, r', n' \models \varphi$.

$\mathsf{EX}\varphi$ means $\neg\mathsf{AX}\neg\varphi$. $\mathcal{T} \models \varphi$ means that for any run r of $\mathcal{T}$, there is $\mathcal{T}, r \models \varphi$, where $\mathcal{T}, r \models \varphi$ means $\mathcal{T}, r, 0 \models \varphi$.

Propositional Agent Abstraction

For our purposes, we focus on *propositional agent abstraction* over LTSs, concerning model-level abstractions instead of theory-level abstractions in agent abstraction [Banihashemi, Giacomo, and Lespérance 2017]. We assume that there are two levels of languages. Let Δ_l be the low-level atoms and $\mathcal{A}_l$ be the low-level action terms; and let Δ_h be the high-level atoms and $\mathcal{A}_h$ be the high-level action terms. We say that an atom or an action term is grounded if its parameters do not contain variables, e.g., $on(B_1, B_2)$ is a grounded atom while $on(B_1, x)$ is not, as x is a variable. In the followings, we assume that atoms and action terms are grounded. A loop-free program δ is defined as follows:

$$\delta ::= \alpha \mid \varphi? \mid (\delta_1; \delta_2) \mid (\delta_1 | \delta_2),$$

where α is an action term; φ is a logical formula, and $\varphi?$ is a test action testing whether φ holds; $(\delta_1; \delta_2)$ means the sequential composition of δ_1 and δ_2; $(\delta_1 | \delta_2)$ means the non-deterministic choice between δ_1 and δ_2.

The key component of propositional agent abstraction is a *propositional refinement mapping*, which maps high-level atoms to low-level formulas, and associates high-level actions to low-level loop-free programs. Formally,

Definition 2. Given a set Φ of formulas and a set Υ of loop-free programs over Δ_l and $\mathcal{A}_l$, a propositional refinement mapping m is a bijection function such that

- for any $f_h \in \Delta_h$, $m(f_h) = \phi$, where $\phi \in \Phi$;
- for any $a_h \in \mathcal{A}_h$, $m(a_h) = \delta$, where $\delta \in \Upsilon$.

Example 1. Suppose that there are two blocks named B_1 and B_2. The refinement mapping in the Introduction is

$m(move\_to\_table) = \delta$, where δ is the following program:

$$\neg ontable(B_1)?; unstack(B_1, B_2); putdown(B_1)$$
$$|\neg ontable(B_2)?; unstack(B_2, B_1); putdown(B_2).$$

$m(all\_ontable) = ontable(B_1) \wedge ontable(B_2)$.

A propositional refinement mapping relates a high-level LTS to a low-level LTS as follows. First, the abstraction relation between high-level states and low-level states is characterized by *m-isomorphic*. Let υ be a function that maps grounded action terms and grounded atoms to actions and propositions in an LTS. We have:

Definition 3. Given a propositional refinement mapping m, a function υ and two LTSs $\mathcal{T}_h$ and $\mathcal{T}_l$, we say that a state $q_h \in Q_h$ is m-isomorphic to a state $q_l \in Q_l$, written $(\mathcal{T}_h, q_h) \sim_m (\mathcal{T}_l, q_l)$, if for any $f_h \in \Delta_h$,

$$\mathcal{T}_h, \upsilon, q_h \models f_h \text{ iff } \mathcal{T}_l, \upsilon, q_l \models m(f_h).$$

The abstraction relation between a high-level LTS and a low-level LTS is then characterized by a notion called *m-bisimulation*, based on m-isomorphic and the semantics of loop-free programs over LTSs.

Definition 4. Given an LTS $\mathcal{T}$ and a loop-free program δ, the semantics of δ is specified by $\mathcal{Q}^v(\mathcal{T}, \delta)$, which is defined inductively:

- $\mathcal{Q}^v(\mathcal{T}, \alpha) \doteq \{(q, q') \mid (q, v(\alpha), q') \in Tr\}$;
- $\mathcal{Q}^v(\mathcal{T}, \varphi?) \doteq \{(q, q) \mid \mathcal{T}, v, q \models \varphi\}$;
- $\mathcal{Q}^v(\mathcal{T}, \delta_1|\delta_2) \doteq \mathcal{Q}^v(\mathcal{T}, \delta_1) \cup \mathcal{Q}^v(\mathcal{T}, \delta_2)$;
- $\mathcal{Q}^v(\mathcal{T}, \delta_1; \delta_2) \doteq \{(q, q'') \mid$ there exists q' s.t. $(q, q') \in \mathcal{Q}^v(\mathcal{T}, \delta_1)$ and $(q', q'') \in \mathcal{Q}^v(\mathcal{T}, \delta_2)\}$.

We assume that programs in a refinement mapping m are *disjoint*, i.e., $\mathcal{Q}^v(\mathcal{T}, \delta) \cap \mathcal{Q}^v(\mathcal{T}, \delta') = \emptyset$, for any $\delta, \delta' \in m$.

Definition 5. Given a function v, a relation $\boldsymbol{B} \subseteq Q_h \times Q_l$ is an m-bisimulation relation between $\mathcal{T}_h$ and $\mathcal{T}_l$ if for any $\langle q_h, q_l\rangle \in \boldsymbol{B}$ implies:

- $(\mathcal{T}_h, q_h) \sim_m (\mathcal{T}_l, q_l)$;
- for any $a_h \in \mathcal{A}_h$, if there exists q_h' such that $(q_h, q_h') \in \mathcal{Q}^v(\mathcal{T}_h, a_h)$, then there exists q_l' such that $(q_l, q_l') \in \mathcal{Q}^v(\mathcal{T}_l, m(a_h))$ and $\langle q_h', q_l'\rangle \in \boldsymbol{B}$;
- for any $a_h \in \mathcal{A}_h$, if there exists q_l' such that $(q_l, q_l') \in \mathcal{Q}^v(\mathcal{T}_l, m(a_h))$, then there exists q_h' such that $(q_h, q_h') \in \mathcal{Q}^v(\mathcal{T}_h, a_h)$ and $\langle q_h', q_l'\rangle \in \boldsymbol{B}$.

If $\langle q_0^h, q_0^l\rangle \in \boldsymbol{B}$, where $\boldsymbol{B}$ is an m-bisimulation relation between $\mathcal{T}_h$ and $\mathcal{T}_l$, then $\mathcal{T}_h$ is m-bisimilar to $\mathcal{T}_l$, written $\mathcal{T}_h \sim_m \mathcal{T}_l$.

Intuitively, if $\mathcal{T}_h$ is m-bisimilar to $\mathcal{T}_l$, then $\mathcal{T}_h$ and $\mathcal{T}_l$ behave the same if $\mathcal{T}_l$ is viewed in an abstract way with the refinement mapping m, by taking low-level programs as high-level actions and low-level formulas as high-level atoms.

Abstraction for LTSs

In this section, we introduce a general abstraction framework for LTSs, and show that propositional agent abstraction can be characterized by it.

We first introduce some notations. Given a function f, we use $dom(f)$ to denote the domain of f, and $range(f)$ to denote the range of f. A *procedure* is a finite sequence of actions. Given an LTS, two states q, q' and a procedure $\tau = a_i; \ldots; a_j$, we use $q \xrightarrow{\tau} q'$ to denote that q' is *reachable* from q via procedure τ, i.e., there exist $q_i, \ldots, q_j, q_{j+1}$ s.t. for each $k \in [i, j]$, there is $(q_k, a_k, q_{k+1}) \in Tr$ and $q_i = q$ and $q_{j+1} = q'$. We call such $q \xrightarrow{\tau} q'$ a *path*, and if q is q_0, then we call such procedure τ a *trace*.

We introduce two notions called state abstractions and procedure abstractions.

Definition 6. Given two sets Q and $\widehat{Q}$ of states, a *state abstraction* is a function $\lambda : Q \to \widehat{Q}$.

Definition 7. Given a set Π of procedures, and a set Act of actions, a *procedure abstraction* is a function $\pi : \Pi \to Act$.

We then introduce abstractions for LTSs

Definition 8. Given an LTS $\mathcal{T}$, an abstraction for $\mathcal{T}$ is a tuple $(\widehat{\mathcal{T}}, \lambda, \pi)$, where $\widehat{\mathcal{T}}$ is an LTS, λ is a state abstraction and π is a procedure abstraction.

Inspired by sound abstraction and complete abstraction for situation calculus models [Cui, Liu, and Luo 2021], we define sound abstraction and complete abstraction for LTSs.

Definition 9. An abstraction $(\widehat{\mathcal{T}}, \lambda, \pi)$ for $\mathcal{T}$ is *sound* if there exists a relation $\boldsymbol{B}$ over $Q \times \widehat{Q}$ s.t. $\langle q_0, \widehat{q_0}\rangle \in \boldsymbol{B}$ and

- $\langle q, \widehat{q}\rangle \in \boldsymbol{B}$ implies $\lambda(q) = \widehat{q}$;
- If $\langle q, \widehat{q}\rangle \in \boldsymbol{B}$ and there exist $a \in \widehat{Act}$ and $\widehat{q}' \in \widehat{Q}$ s.t. $\widehat{q} \xrightarrow{a} \widehat{q}'$, then there exist $\tau \in dom(\pi)$ and $q' \in Q$ s.t. $q \xrightarrow{\tau} q'$ and $\pi(\tau) = a$ and $\langle q', \widehat{q}'\rangle \in \boldsymbol{B}$.

Sound abstraction guarantees that if there is a trace ending at a state of the abstract LTS, then there exists a *refined* trace ending at a *refined* state of the concrete LTS.

Formally, if notation $\pi^\star(\tau, \widehat{\tau})$ denotes that $\widehat{\tau}$ is an abstract procedure of procedure τ, i.e., τ can be partitioned into $\tau_1; \ldots; \tau_n$ s.t. $\pi(\tau_1) = a_1, ..., \pi(\tau_n) = a_n$ and thus $\widehat{\tau} = a_1; \ldots; a_n$, then we have:

Proposition 1. *If $(\widehat{\mathcal{T}}, \lambda, \pi)$ is sound abstraction for $\mathcal{T}$, then for any state $\widehat{q}$ and procedure $\widehat{\tau}$ s.t. $\widehat{q_0} \xrightarrow{\widehat{\tau}} \widehat{q}$, there exist state q and procedure τ s.t. $q_0 \xrightarrow{\tau} q$ and $\lambda(q) = \widehat{q}$ and $\pi^\star(\tau, \widehat{\tau})$.*

Proof (sketch). We prove by induction on the length of procedure $\widehat{\tau}$. □

Definition 10. An abstraction $(\widehat{\mathcal{T}}, \lambda, \pi)$ for $\mathcal{T}$ is *complete* if there exists a relation $\boldsymbol{B}$ over $Q \times \widehat{Q}$ s.t. $\langle q_0, \widehat{q_0}\rangle \in \boldsymbol{B}$ and

- $\langle q, \widehat{q}\rangle \in \boldsymbol{B}$ implies $\lambda(q) = \widehat{q}$;
- If $\langle q, \widehat{q}\rangle \in \boldsymbol{B}$ and there exist $\tau \in dom(\pi)$ and $q' \in Q$ s.t. $q \xrightarrow{\tau} q'$, then there exist $a \in \widehat{Act}$ and $\widehat{q}' \in \widehat{Q}$ s.t. $\widehat{q} \xrightarrow{a} \widehat{q}'$ and $\pi(\tau) = a$ and $\langle q', \widehat{q}'\rangle \in \boldsymbol{B}$.

Complete abstraction guarantees that given a trace ending at a state of the concrete LTS, if this trace is *abstractable*, then there exists an abstract trace ending at the abstract state of the abstract LTS. Formally,

Proposition 2. *If $(\widehat{\mathcal{T}}, \lambda, \pi)$ is complete abstraction for $\mathcal{T}$, then for any state q and procedure τ s.t. $q_0 \xrightarrow{\tau} q$, if there exists a procedure $\widehat{\tau}$ s.t. $\pi^\star(\tau, \widehat{\tau})$, then there exists state $\widehat{q}$ s.t. $\lambda(q) = \widehat{q}$ and $\widehat{q_0} \xrightarrow{\widehat{\tau}} \widehat{q}$.*

Abstractions can be deterministic and non-deterministic.

Definition 11. An abstraction $(\widehat{\mathcal{T}}, \lambda, \pi)$ for $\mathcal{T}$ is deterministic if there exists a relation $\boldsymbol{B}$ over $Q \times \widehat{Q}$ s.t. $\langle q_0, \widehat{q_0}\rangle \in \boldsymbol{B}$ and

- $\langle q, \widehat{q}\rangle \in \boldsymbol{B}$ implies $\lambda(q) = \widehat{q}$;
- If $\langle q, \widehat{q}\rangle \in \boldsymbol{B}$ and there exist $a \in \widehat{Act}$ and $\widehat{q}' \in \widehat{Q}$ s.t. $\widehat{q} \xrightarrow{a} \widehat{q}'$, then for any $\tau \in dom(\pi)$ and $q' \in Q$ s.t. $q \xrightarrow{\tau} q'$, if $\pi(\tau) = a$, then $\langle q', \widehat{q}'\rangle \in \boldsymbol{B}$.

If an abstraction is deterministic, then an abstract trace can only end at an unique final state:

Proposition 3. *If $(\widehat{\mathcal{T}}, \lambda, \pi)$ for $\mathcal{T}$ is deterministic, then if there exist state $\widehat{q}$ and procedure $\widehat{\tau}$ s.t. $\widehat{q_0} \xrightarrow{\widehat{\tau}} \widehat{q}$, then for any $\widehat{q}'$ s.t. $\widehat{q_0} \xrightarrow{\widehat{\tau}} \widehat{q}'$, we have $\widehat{q}' = \widehat{q}$.*

Proof (sketch). We prove by induction on the length of procedure $\widehat{\tau}$. □

In the followings, we show that propositional agent abstraction is a strong version of non-deterministic, sound and complete abstraction (NDeSCA), which we call NDeSCA$^+$. Similarly, we introduce a strong version of deterministic, sound and complete abstraction (DeSCA) called DeSCA$^+$.

Definition 12. An abstraction $(\widehat{\mathcal{T}}, \lambda, \pi)$ for $\mathcal{T}$ is NDeSCA$^+$ if it is sound abstraction with some relation $\boldsymbol{B}$ and complete abstraction with some $\boldsymbol{B}'$, and $\boldsymbol{B} = \boldsymbol{B}'$.

NDeSCA$^+$ is stronger than NDeSCA, in that the relations that constitute sound abstraction and complete abstraction are the same in NDeSCA$^+$ while they are not necessarily the same in NDeSCA. Similarly, we have:

Definition 13. An abstraction $(\widehat{\mathcal{T}}, \lambda, \pi)$ for $\mathcal{T}$ is DeSCA$^+$ if it is NDeSCA$^+$ with some relation $\boldsymbol{B}$, and deterministic abstraction with some $\boldsymbol{B}'$, and $\boldsymbol{B} = \boldsymbol{B}'$.

We show that m-bisimilar relation can be characterized by our framework.

Proposition 4. *Given a refinement mapping m and LTSs $\widehat{\mathcal{T}}$ and $\mathcal{T}$, there exist a state abstraction λ and a procedure abstraction π s.t. $\widehat{\mathcal{T}} \sim_m \mathcal{T}$ iff $(\widehat{\mathcal{T}}, \lambda, \pi)$ is NDeSCA$^+$ for $\mathcal{T}$.*

Proof (sketch). We first show that from a refinement mapping m, we can induce a state abstraction λ and a procedure abstraction π as follows. If $(\widehat{\mathcal{T}}, \widehat{q}) \sim_m (\mathcal{T}, q)$, then $\lambda(q) = \widehat{q}$. For any loop-free program $\delta \in m$, if $(q, q') \in \mathcal{Q}^v(\mathcal{T}, \delta)$, and there exists a procedure τ s.t. $q \xrightarrow{\tau} q'$, then $\pi(\tau) = a_\delta$, where a_δ is an abstract action we use for δ. We then show that m-bisimilar coincides with NDeSCA$^+$. □

Finally, we define our verification problems.

Definition 14. NDeSCA$^+$ existence problem is that given an LTS $\mathcal{T}$, a state abstraction λ and a procedure abstraction π, determine if there exists an LTS $\widehat{\mathcal{T}}$ s.t. $(\widehat{\mathcal{T}}, \lambda, \pi)$ is NDeSCA$^+$ for $\mathcal{T}$.

Definition 15. DeSCA$^+$ existence problem is that given an LTS $\mathcal{T}$, a state abstraction λ and a procedure abstraction π, determine if there exists an LTS $\widehat{\mathcal{T}}$ s.t. $(\widehat{\mathcal{T}}, \lambda, \pi)$ is DeSCA$^+$ for $\mathcal{T}$.

Verifying the Existence of Abstraction

In this section, we show how to solve NDeSCA$^+$ and DeSCA$^+$ existence problem via CTLK model checking.

We first investigate necessary and sufficient conditions under which there exists complete abstraction and there exists sound abstraction, given an LTS, a state abstraction λ, and a procedure abstraction π.

The conditions of the existence of complete abstraction make use of a notion called *π-reachable states*.

Definition 16. Given an LTS $\mathcal{T}$ and a procedure abstraction π, the π-reachable states of $\mathcal{T}$, written $Q^\pi_\mathcal{T}$, is defined inductively:

- $q_0 \in Q^\pi_\mathcal{T}$;
- for any $q \in Q^\pi_\mathcal{T}$ and for any $\tau \in dom(\pi)$, if there exists state q' s.t. $q \xrightarrow{\tau} q'$, then $q' \in Q^\pi_\mathcal{T}$.

We show that all the π-reachable states should be abstracted for the existence of complete abstraction. Formally, let q be a state and λ be a state abstraction. We use $\lambda(q) = \perp$ to denote that $\lambda(q)$ is not defined. Then,

Proposition 5. *Given an LTS $\mathcal{T}$, a state abstraction λ and a procedure abstraction π, if there exists complete abstraction for $\mathcal{T}$, then $\forall q \in Q^\pi_\mathcal{T}.\lambda(q) \neq \perp$.*

Proof (sketch). We prove by contradiction. Suppose that there exist complete abstraction and $q \in Q^\pi_\mathcal{T}$ s.t. $\lambda(q) = \perp$. We show that by establishing a relation $\boldsymbol{B}$ for complete abstraction, as $q \in Q^\pi_\mathcal{T}$, $\langle q, \widehat{q}\rangle$ must be in $\boldsymbol{B}$ for a certain $\widehat{q}$. It follows that $\lambda(q) = \widehat{q}$, which comes to a contradiction. □

With the π-reachable states, we show how to obtain complete abstraction. We make use of the following notations.

$$abs(q \xrightarrow{\tau} q', \widehat{q} \xrightarrow{a} \widehat{q}') \doteq q \xrightarrow{\tau} q' \text{ and } \lambda(q) = \widehat{q} \text{ and } \lambda(q') = \widehat{q}' \text{ and } \pi(\tau) = a,$$

which means that $q \xrightarrow{\tau} q'$ is a path *abstractable* to $\widehat{q} \xrightarrow{a} \widehat{q}'$. Given a state set Q, if there exists $q \xrightarrow{\tau} q'$ s.t. $abs(q \xrightarrow{\tau} q', \widehat{q} \xrightarrow{a} \widehat{q}')$ holds, then we call such $\widehat{q} \xrightarrow{a} \widehat{q}'$ an *$\exists$-abstract-edge* wrt Q, written $\mathcal{E}^\exists_Q(\widehat{q} \xrightarrow{a} \widehat{q}')$.

Proposition 6. *Given an LTS $\mathcal{T}$, a state abstraction λ, and a procedure abstraction π, if $\forall q \in Q^\pi_\mathcal{T}.\lambda(q) \neq \perp$, then there exists complete abstraction for $\mathcal{T}$.*

Proof (sketch). We show how to build an LTS $\widehat{\mathcal{T}}$ that is complete abstraction for $\mathcal{T}$. The intuition is that we let states in $\widehat{\mathcal{T}}$ be the states abstracted from $Q^\pi_\mathcal{T}$, the π-reachable states of $\mathcal{T}$. We let transitions in $\widehat{\mathcal{T}}$ be $\exists$-abstract-edges wrt $Q^\pi_\mathcal{T}$. We can show that $(\widehat{\mathcal{T}}, \lambda, \pi)$ is complete abstraction for $\mathcal{T}$ by establishing a relation $\boldsymbol{B}$ inductively. □

To build sound abstraction, we introduce a similar concept called *$\forall$-abstract-edge* wrt Q, written $\mathcal{E}^\forall_Q(\widehat{q} \xrightarrow{a} \widehat{q}')$.

$$\mathcal{E}^\forall_Q(\widehat{q} \xrightarrow{a} \widehat{q}') \doteq \forall q \in Q, \exists \tau \in dom(\pi), \exists q' \in Q. \; if \lambda(q) = \widehat{q}, \text{ then } abs(q \xrightarrow{\tau} q', \widehat{q} \xrightarrow{a} \widehat{q}'),$$

which means that for any state q abstractable to $\widehat{q}$, there exists a path $q \xrightarrow{\tau} q'$ abstractable to $\widehat{q} \xrightarrow{a} \widehat{q}'$.

Note however that, it is trivial to show that sound abstraction exists when $\lambda(q_0) \neq \perp$, as one can construct a trivial abstract LTS which contains only a state $\lambda(q_0)$.

With the above results, we show the necessary and sufficient conditions for the existence of NDeSCA$^+$ and the existence of DeSCA$^+$.

Theorem 1. *Given an LTS $\mathcal{T}$, a state abstraction λ, and a procedure abstraction π, there exists an LTS $\widehat{\mathcal{T}}$ s.t. $(\widehat{\mathcal{T}}, \lambda, \pi)$ is NDeSCA$^+$ for $\mathcal{T}$ iff the following two conditions hold:*

- *$\forall q \in Q^\pi_\mathcal{T}.\lambda(q) \neq \perp$;*
- *For any $\widehat{q}, a, \widehat{q}'$, if $\mathcal{E}^\exists_{Q^\pi_\mathcal{T}}(\widehat{q} \xrightarrow{a} \widehat{q}')$, then $\mathcal{E}^\forall_{Q^\pi_\mathcal{T}}(\widehat{q} \xrightarrow{a} \widehat{q}')$.*

Proof (sketch). We first prove the **if-direction**. By Prop. 6, there exists complete abstraction. We show that this complete abstraction is also sound abstraction by the definition of $\mathcal{E}^{\forall}_{Q^{\pi}_{\mathcal{T}}}(\widehat{q} \xrightarrow{a} \widehat{q}')$. We then prove the **only-if-direction**. By Prop. 5, the first condition is necessary. We prove that the second condition is necessary by contradiction. Suppose that there exist $\widehat{q}, a, \widehat{q}'$ s.t. $\mathcal{E}^{\exists}_{Q^{\pi}_{\mathcal{T}}}(\widehat{q} \xrightarrow{a} \widehat{q}')$ holds while $\mathcal{E}^{\forall}_{Q^{\pi}_{\mathcal{T}}}(\widehat{q} \xrightarrow{a} \widehat{q}')$ does not hold. It follows that there exists $q \in Q^{\pi}_{\mathcal{T}}$ s.t. $abs(q \xrightarrow{\tau} q', \widehat{q} \xrightarrow{a} \widehat{q}')$ does not hold. Since $\langle q, \widehat{q} \rangle$ must be in $\boldsymbol{B}$, we fail to obtain $\boldsymbol{B}$ for sound abstraction, which comes to a contradiction. □

Intuitively, the above theorem says that if the abstract LTS built via ∃-abstract-edges wrt the π-reachable states results in the same abstract LTS built via ∀-abstract-edges wrt the π-reachable states, then $\texttt{NDeSCA}^+$ exists.

Additionally, to see whether an abstraction is deterministic, we check whether at any π-reachable state, if two procedures are bound to the same abstract action, then their resulting states are bound to the same abstract state.

Theorem 2. *Given an LTS $\mathcal{T}$, a state abstraction λ, and a procedure abstraction π, there exists an LTS $\widehat{\mathcal{T}}$ s.t. $(\widehat{\mathcal{T}}, \lambda, \pi)$ is $\texttt{DeSCA}^+$ for $\mathcal{T}$ iff the following two conditions hold:*

- *$(\widehat{\mathcal{T}}, \lambda, \pi)$ is $\texttt{NDeSCA}^+$ for $\mathcal{T}$;*
- *for any q, q', $q'' \in Q^{\pi}_{\mathcal{T}}$ and any $\tau, \omega \in dom(\pi)$ s.t., $q \xrightarrow{\tau} q'$ and $q \xrightarrow{\omega} q''$, if $\pi(\tau) = \pi(\omega)$, then $\lambda(q') = \lambda(q'')$.*

Proof (sketch). By Theorem 1, we only need to prove that the second condition holds iff $(\widehat{\mathcal{T}}, \lambda, \pi)$ is deterministic. We first prove the **if-direction**. We show that $(\widehat{\mathcal{T}}, \lambda, \pi)$ is also deterministic (Def. 11) by the second condition. We then prove the **only-if-direction** by contradiction. Suppose that there exist q, q', $q'' \in Q^{\pi}_{\mathcal{T}}$ and $\tau, \omega \in dom(\pi)$ s.t. $q \xrightarrow{\tau} q'$, $q \xrightarrow{\omega} q''$, $\pi(\tau) = \pi(\omega)$ and $\lambda(q') \neq \lambda(q'')$. There must exist $\widehat{q}$ s.t. $\langle q, \widehat{q} \rangle \in \boldsymbol{B}$. Since $q \xrightarrow{\tau} q'$, suppose that $\pi(\tau) = a$ and $\lambda(q') = \widehat{q}'$, by Def. 10, we have $\widehat{q} \xrightarrow{a} \widehat{q}'$. Since $q \xrightarrow{\omega} q''$, $\pi(\omega) = \pi(\tau) = a$, by Def. 11, we have $\lambda(q') = \widehat{q}' = \lambda(q'')$, which violates $\lambda(q') \neq \lambda(q'')$. □

In the following, we reduce $\texttt{DeSCA}^+$ and $\texttt{NDeSCA}^+$ existence problems to CTLK model-checking problems. We assume that a state abstraction λ is complete, i.e., $\forall q.\lambda(q) \neq \bot$. The intuition for reduction is to make use of indistinguishability. We say that two concrete states are indistinguishable if they are abstracted to the same state; two procedures are indistinguishable if they are abstracted to the same action; two paths are indistinguishable if both their procedures and ending states are indistinguishable.

We first induce an ETS from an LTS by

- focusing on the π-reachable states and their transitions;
- introducing propositional variables for abstract actions and abstract states;
- labelling states to indicate indistinguishable paths;
- introducing an indistinguishable relation over states.

Definition 17. Let $\mathcal{T} = (Q, q_0, Act, Tr, \Delta, l)$ be an LTS, λ be a state abstraction, and π be a procedure abstraction. The induced ETS of $\mathcal{T}$ wrt λ and π, written $\mathcal{T}_{\lambda,\pi}$, is a tuple $(Q', q'_0, R', \Delta', l', \sim)$, where

- $Q' = Q^{\pi}_{\mathcal{T}}$ and $q'_0 = q_0$;
- $R' = \{(q, q') \mid q \xrightarrow{\tau} q' \text{ for } q, q' \in Q', \tau \in dom(\pi)\}$;
- $\Delta' = \{p_{\widehat{a}} \mid \widehat{a} \in range(\pi)\} \cup \{p_{\widehat{s}} \mid \widehat{s} \in range(\lambda)\}$;
- $l'(q) = \{p_{\widehat{a}} \mid \text{there exist } s \in Q', \tau \in dom(\pi) \text{ and } \widehat{a} \in range(\pi) \text{ s.t. } s \xrightarrow{\tau} q \text{ and } \pi(\tau) = \widehat{a}\ \} \cup \{p_{\lambda(q)}\}$;
- $\sim = \{(q, q') \mid \lambda(q) = \lambda(q') \text{ for } q, q' \in Q'\}$.

With the induced ETS $\mathcal{T}_{\lambda,\pi}$, to decide whether $\texttt{NDeSCA}^+$ exists for $\mathcal{T}$, we can check whether the following condition holds: for any state q of $\mathcal{T}_{\lambda,\pi}$, if there exists a path starting from q, then for any its indistinguishable state q', there exists an indistinguishable path starting from q'. This condition can be formalized by the logic CTLK.

Theorem 3. *Given an LTS $\mathcal{T}$, a state abstraction λ and a procedure abstraction π, there exists an LTS $\widehat{\mathcal{T}}$ s.t. $(\widehat{\mathcal{T}}, \lambda, \pi)$ is $\texttt{NDeSCA}^+$ for $\mathcal{T}$ iff the induced ETS $\mathcal{T}_{\lambda,\pi}$ satisfies that for any $\widehat{a} \in range(\pi)$ and $\widehat{s} \in range(\lambda)$,*

$$\mathcal{T}_{\lambda,\pi} \models \mathsf{AG}(\mathsf{EX}(p_{\widehat{a}} \wedge p_{\widehat{s}}) \supset \mathbf{K}\mathsf{EX}(p_{\widehat{a}} \wedge p_{\widehat{s}})).$$

Proof (sketch). To prove the **if-direction**, by the semantics of CTLK over LTSs and the definition of $\mathcal{T}_{\lambda,\pi}$, we can show that it satisfies the conditions for $\texttt{NDeSCA}^+$. The **only-if-direction** can be proved by contradiction. If the checking fails, by the definition of $\mathcal{T}_{\lambda,\pi}$, we can show that it violates the conditions for $\texttt{NDeSCA}^+$. □

Theorem 4. *Given an LTS $\mathcal{T}$, a state abstraction λ and a procedure abstraction π, there is an LTS $\widehat{\mathcal{T}}$ s.t. $(\widehat{\mathcal{T}}, \lambda, \pi)$ is $\texttt{DeSCA}^+$ for $\mathcal{T}$ iff the induced ETS $\mathcal{T}_{\lambda,\pi}$ satisfies that for any $\widehat{a} \in range(\pi)$ and $\widehat{s} \in range(\lambda)$,*

$$\mathcal{T}_{\lambda,\pi} \models \mathsf{AG}(\mathsf{EX}(p_{\widehat{a}} \wedge p_{\widehat{s}}) \supset \mathbf{K}\mathsf{EX}(p_{\widehat{a}} \wedge p_{\widehat{s}}));$$
$$\mathcal{T}_{\lambda,\pi} \models \mathsf{AG}(\mathsf{EX}(p_{\widehat{a}} \wedge p_{\widehat{s}}) \supset \mathsf{AX}(p_{\widehat{a}} \supset p_{\widehat{s}})).$$

Proof (sketch). By Theorem 2 and Theorem 3, we only need to prove that for any $\widehat{a} \in range(\pi)$ and $\widehat{s} \in range(\lambda)$, $\mathcal{T}_{\lambda,\pi} \models \mathsf{AG}(\mathsf{EX}(p_{\widehat{a}} \wedge p_{\widehat{s}}) \supset \mathsf{AX}(p_{\widehat{a}} \supset p_{\widehat{s}}))$ iff the second condition in Theorem 2 holds. It follows by the semantics of CTLK. □

As the complexity of CTLK model checking is PTIME-complete [Raimondi 2006] wrt the model size and formula size, by Def. 17 and Theorem 3&4, we have:

Corollary 1. *$\texttt{DeSCA}^+$ existence problem and $\texttt{NDeSCA}^+$ existence problem can be solved in* PTIME.

Automated Verification

In this section, we show how to verify propositional agent abstraction in the setting of propositional STRIPS planning [Bylander 1994]: Signature Σ is a set of propositional variables used in a propositional language. A *literal* is either a propositional variable p or its negation $\neg p$. Let Θ be a set of literals. $\widetilde{\Theta}$ denotes $\{\neg l \mid l \in \Theta\}$. A *cube* is a set of literals,

understood as the conjunction of them. An action a is characterized by a tuple $(\mathit{pre}_a, \mathit{eff}_a)$, where pre_a is the precondition and eff_a is the effects, which are cubes. An action a is applicable in state s if $s \models \mathit{pre}_a$. If an action a is applicable, executing a in state s results in a successor state s', which is the unique state s.t. $s' \models \mathit{eff}_a$ and for any $l \notin \mathit{eff}_a \cup \widetilde{\mathit{eff}}_a$, there is $s' \models l$ iff $s \models l$. A propositional STRIPS planning domain D is a tuple $(X, s_I, \mathcal{A})$, where X is s set of propositional variables, s_I is the initial state, and $\mathcal{A}$ is a set of actions. Given a propositional STRIPS planning domain D, it is trivial to induce an LTS, which we denote as $\mathcal{T}_D$.

Our problem is defined as follows. Given a propositional STRIPS planning domain $D = (X, s_I, \mathcal{A})$ and a refinement mapping m, which is devised over an abstract language and the concrete elements X and $\mathcal{A}$, we decide if there is an abstract planning domain D' s.t. the induced LTSs $\mathcal{T}_{D'}$ and $\mathcal{T}_D$ satisfy $\mathcal{T}_{D'} \sim_m \mathcal{T}_D$. As we have shown that m-bisimilar relation can be characterised by $\mathtt{NDeSCA}^+$, and verifying the existence of $\mathtt{NDeSCA}^+$ needs the induced ETS, our problem lies in how to construct the ETS from D and m. This construction is symbolic, as both D and m are symbolic.

Thus our verification process first generates a symbolic representation of ETS in Def. 17, given D and m, and then performs symbolic model checking against CTLK specifications by Theorem 3.

To get the transition relation of a symbolic ETS, we treat a loop-free program occurring in refinement mapping m as an action in planning, thus we first show how to obtain the precondition and effects of a loop-free program. Similar to propositional formulas, we introduce a normal form of a program called NRSP. A program is of the form NRSP if it is the non-deterministic choice of *restricted sequential programs* (RSPs). A sequential program is a program resulting from the sequential composition of actions. A sequential program is restricted if the formula ϕ of any test action is a conjunctive clause, i.e., the conjunction of literals. Given a program δ, we can obtain an equivalent program of the form NRSP as follows: we first obtain an equivalent δ' from δ by translating each formula ϕ in δ to an equivalent DNF formula ϕ' and splitting ϕ' s.t. formulas in δ' are conjunctive clauses. For instance, program $(p_1 \wedge p_2 \vee p_3)?; a$ is rewritten to $(p_1 \wedge p_2?; a)|(p_3?; a)$. We then translate δ' to an equivalent program in the form of NRSP. For instance, program $(a|b); c$ is rewritten to $(a; c)|(b; c)$. We use $\mathsf{RSP}(\delta)$ to denote the set of RSPs with δ being of the form NRSP. Hence we can focus on obtaining preconditions and effects of RSPs.

Before doing so, we introduce some notations, and operations over sets of literals. Given a sequential program δ and a natural number i, notation $\delta[i]$ denotes the ith action in program δ. Notation $\top$ denotes the set containing no literals, meaning that all states are possible, and $\bot$ represents the set containing all the literals, meaning that no state is possible. We have $\bot \supseteq \Delta \supseteq \top$, for any set Δ of literals. Given two sets Γ, Θ of literals, $\Gamma \ominus \Theta$ is a new set of literals such that if there is no literal l s.t. $l \in \Gamma$ and $\neg l \in \Theta$, then $\Gamma \ominus \Theta = \Gamma \setminus \Theta$; otherwise $\Gamma \ominus \Theta = \bot$. $\Gamma \uplus \Theta$ is a new set of literals such that if there is no literal l s.t. $l \in \Gamma$ and $\neg l \in \Theta$, then $\Gamma \uplus \Theta = \Gamma \cup \Theta$; otherwise $\Gamma \uplus \Theta = \bot$.

Definition 18. Given a RSP δ, the precondition of δ, written pre^*_δ, is $\mathit{pre}(\delta, \top)$, where $\mathit{pre}(\delta, \Delta)$ is defined as:

- $\mathit{pre}(a, \Delta) \doteq \mathit{pre}_a \uplus (\Delta \ominus \mathit{eff}_a)$, where a is an action;
- $\mathit{pre}(\phi?, \Delta) \doteq L(\phi) \uplus \Delta$, where $L(\phi)$ denotes literals in ϕ;
- $\mathit{pre}(\delta_1; \delta_2, \Delta) \doteq \mathit{pre}(\delta_1, \mathit{pre}(\delta_2, \Delta))$,

Proposition 7. *Given a RSP δ, a function υ, and a state q in an LTS $\mathcal{T}$, we have that $\mathcal{T}, \upsilon, q \models \mathrm{pre}^*_\delta$ iff there exists state q' s.t. $(q, q') \in \mathcal{Q}^\upsilon(\mathcal{T}, \delta)$.*

Proof (sketch). We first prove that $\mathcal{T}, \upsilon, q \models \mathit{pre}(\delta, \Delta)$ iff there exists state q' s.t. $(q, q') \in \mathcal{Q}^\upsilon(\mathcal{T}, \delta)$ and $\mathcal{T}, \upsilon, q' \models \Delta$. We prove that by structural induction on δ. When Δ is $\top$, it follows that the proposition holds. □

Definition 19. Given a RSP δ, the effects of δ, written eff^*_δ, are $\mathit{eff}(\top, \delta)$, where $\mathit{eff}(\Delta, \delta)$ is defined as:

- $\mathit{eff}(\Delta, a) \doteq \Delta \setminus \{l, \neg l \mid l \in \mathit{eff}_a\} \cup \mathit{eff}_a$;
- $\mathit{eff}(\Delta, \phi?) \doteq \Delta$;
- $\mathit{eff}(\Delta, \delta_1; \delta_2) \doteq \mathit{eff}(\mathit{eff}(\Delta, \delta_1), \delta_2)$.

Proposition 8. *Given a RSP δ, a function υ, and a state q in an LTS $\mathcal{T}$, if there is state q' s.t. $(q, q') \in \mathcal{Q}^\upsilon(\mathcal{T}, \delta)$, then $\mathcal{T}, \upsilon, q' \models \mathrm{eff}^*_\delta$ and for any $l \notin \mathrm{eff}^*_\delta \cup \widetilde{\mathrm{eff}}^*_\delta$, $q' \models l$ iff $q \models l$.*

Proof (sketch). We first prove that for any $l \in \Delta$, we have $l \in \mathit{eff}(\Delta, \delta)$ or $\neg l \in \mathit{eff}(\Delta, \delta)$ by structural induction on δ. We then prove a stronger proposition that if $\mathcal{T}, \upsilon, q \models \Delta$ and there exists state q' s.t. $(q, q') \in \mathcal{Q}^\upsilon(\mathcal{T}, \delta)$, then $\mathcal{T}, \upsilon, q' \models \mathit{eff}(\Delta, \delta)$ and for any $l \notin (\mathit{eff}(\Delta, \delta) \setminus \Delta) \cup (\widetilde{\mathit{eff}}(\Delta, \delta) \setminus \widetilde{\Delta})$, there is $q' \models l$ iff $q \models l$ by structural induction. When Δ is $\top$, it follows that the original proposition holds. □

We then show how to induce a symbolic representation of an ETS, which can be viewed as a result of an agent executing a physical action and a sensing action alternatively in an environment. A physical action has the same precondition and effects as a low-level RSP of a loop-free program of the refinement mapping. The only sensing action senses the truth values of low-level properties represented by formulas of the refinement mapping.

Definition 20. Given a propositional STRIPS planning domain $D = (X, s_I, \mathcal{A})$ and a propositional refinement mapping m, a symbolic epistemic transition system S is a tuple $\langle X', s'_I, \mathcal{A}', \mathcal{A}_o \rangle$, where

- $X' = X \cup X_\mathcal{P} \cup X_\mathcal{F} \cup \{p_o\}$, where $X_\mathcal{P} = \{p_\delta \mid \delta \in range(m)\}$; $X_\mathcal{F} = \{p_\phi \mid \phi \in range(m)\}$;
- $s'_I = s_I \cup \{p_o\}$
- $\mathcal{A}' = \{a_{\delta_i} \mid \delta_i \in \mathsf{RSP}(\delta), \delta \in range(m)\}$, where a_{δ_i} is a physical action determined by $(\mathit{pre}_{a_{\delta_i}}, \mathit{eff}_{a_{\delta_i}})$, where
  - $\mathit{pre}_{a_{\delta_i}} = \mathit{pre}^*_{\delta_i} \cup \{\neg p_o\}$;
  - $\mathit{eff}_{a_{\delta_i}} = \mathit{eff}^*_{\delta_i} \cup \{p_o\} \cup \{p_\delta\} \cup \{\neg p_{\delta'} \mid p_{\delta'} \in X_\mathcal{P} \setminus \{p_\delta\}\}$.
- $\mathcal{A}_o = \{a_o\}$, where a_o is a sensing action determined by $(\mathit{pre}_{a_o}, \mathit{eff}_{a_o})$, where $\mathit{pre}_{a_o} = \{p_o\}$ and $\mathit{eff}_{a_o} = \{\phi \supset p_\phi, \neg\phi \supset \neg p_\phi \mid \phi \in range(m)\} \cup \{\neg p_o\}$.

$X_\mathcal{P}$ records which loop-free program is performed. $X_\mathcal{F}$ evaluates low-level properties. p_o is for the execution alternating between a sensing action and a physical action.

Given a symbolic ETS, it is trivial to derive an ETS. But the induced ETS has spurious states, in that we only need to focus on states resulting from the sensing action. Thus CTLK specifications are that for any $p_\delta \in X_\mathcal{P}$ and $p_\phi \in X_\mathcal{F}$, $\mathsf{AG}(\neg p_o \wedge \mathsf{EXEX}(p_\delta \wedge p_\phi) \supset \mathbf{K}\mathsf{EXEX}(p_\delta \wedge p_\phi))$.

Experiments

To evaluate our approach, we implemented a prototype system called PAAChecker[1] using Python. For CTLK model checking, we make use of the state-of-the-art tool, MCMAS [Lomuscio, Qu, and Raimondi 2017]. The time-out bound is 3600 seconds. All experiments ran on a Linux machine with 2.2 GHz CPU and 96GB memory. To simplify specifications, refinement mappings are first-order and will be propositional via the *grounding* process, given a set of objects. Program $\pi(k).action(k)$ means performing $action$ with a certain object k. Our tested domains are Blocksworld [Slaney and Thiébaux 2001], Robot [Kabanza, Barbeau, and St-Denis 1997] and Childsnack [Vallati et al. 2015].

Blocksworld A robot is re-assembling stackable blocks on a table with unlimited space. The robot can stack a block onto another block, unstack a block from another block, put down a block, or pick up a block from the table. Auxiliary predicates $goal\_ontable$, $goal\_on$, and $done$ related to planning goals are introduced.

We devise a refinement mapping as follows. High-level action $onestep\_move$ is the non-deterministic choice of

- if $\pi(x,y).\neg done(x) \wedge done(y) \wedge goal\_on(x,y)$, then $pickup(x)$ or $\pi(k).unstack(x,k)$, and then $stack(x,y)$
- if $\pi(x).\neg done(x) \wedge \neg on\_table(x)$, then $\pi(k).unstack(x,k)$, and then $putdown(x)$
- if $\pi(x).on\_table(x) \wedge goal\_on\_table(x) \vee \exists y.on(x,y) \wedge goal\_on(x,y) \wedge done(y)$, then $mark\_done(x)$
- if $\forall x.done(x)$, then do nothing.

High-level atom $success$ denotes the disjunction of

- $\exists x,y.\neg done(x) \wedge goal\_on(x,y) \wedge done(y)$
- $\exists x.\neg done(x) \wedge \neg on\_table(x)$
- $\exists x.on\_table(x) \wedge goal\_on\_table(x) \wedge \neg done(x)$
- $\exists x,y.on(x,y) \wedge goal\_on(x,y) \wedge done(y) \wedge \neg done(x)$
- $\forall x.done(x)$.

Robot A robot is picking up objects in their initial locations and placing them in their goal locations. The robot also can open a door, or move from one location to another location given that the door is open. We devise a refinement mapping where 4 high-level actions h_pickup, $move\_release$, $move\_to\_object\ h\_holding$, and 3 high-level atoms $pickable$, $movable$, $success$ are given.

Childsnack A robot is making and serving sandwiches for a group of children in which some are allergic to gluten. The robot can make a sandwich or make a sandwich taking into account that all ingredients are gluten-free. She also can put a sandwich on a tray, move a tray from one place to another or serve sandwiches. We devise a refinement mapping where a high-level action $serve$ and a high-level atom $success$ are given.

[1]It is available at https://github.com/luokailun/PAAChecker

Instance	#✔	#✘	#S	T_{mc}(s)	T(s)
B10	1	0	550	2.0	2.1
B15	1	0	5666	18.9	19.0
B20	1	0	119682	35.8	36.0
B25	1	0	73506	766.0	766.2
B30	1	0	6.6e+06	1691.4	1691.6
B35	1	0	312678	2437.3	2437.7
R04-05	7	5	291	0.5	0.5
R05-10	9	3	86	0.7	0.8
R10-20	7	5	24295	91.9	92.0
R15-30	–	–	–	–	Timeout
R20-40	8	4	273535	448.1	448.2
R25-50	–	–	–	–	Timeout
C04	1	0	132850	37.1	39.2
C06	1	0	2.2e+06	136.3	143.5
C08	1	0	3.3e+06	853.9	862.6
C10	1	0	5.9e+07	1650.4	1676.7
C12	1	0	7.2e+07	1179.2	1208.3
C14	–	–	–	–	Timeout

Table 1: Experimental results

Experimental results are summarized in Table 1, where #✔ (resp. #✘) denotes the number of formulas verified to be true (resp. false); #S and T_{mc} (in seconds) denote the number of reachable states and the time for CTLK model checking; T is the total time for running the system. Six instances of an increasing number of objects of each domain are tested. On an instance of medium size, our system can return a result in a reasonable amount of time. On a large instance, the verification seems too computationally expensive with a complicated refinement mapping. Note however that, a propositional refinement mapping m can be reused for many planning problems. That is, if m is verified to be correct for domain instance D, then it is suitable for any planning problem (D, g), given that the planning goal g is *abstractable* by m. Thus our system is suitable for the scenario where the planning goal is often changed.

Conclusion

We have investigated the existence problem of propositional agent abstraction, given a refinement mapping and a planning domain in the propositional case. We have shown that the problem is reducible to a CTLK model checking problem, which yields an automated approach for verification. We implemented a prototype system, and the experimental results showed that our system can solve domains of medium size in a reasonable amount of time.

In the future, to solve larger domains, we would like to exploit compacted representation such as SAS^+ [Helmert 2009], and consider planning-goal-oriented abstraction by relaxing the requirement of complete abstraction. Based on our work, we would like to explore the synthesis of abstract planning domains and the synthesis of refinement mappings.

Acknowledgments

We thank the anonymous reviewers for helpful comments. We acknowledge support from the National Key Research and Development Program of China under Grant No.2021YFB3101300 and the Natural Science Foundation of China under Grant No.62206055.

References

Baier, J. A.; Fritz, C.; and McIlraith, S. A. 2007. Exploiting Procedural Domain Control Knowledge in State-of-the-Art Planners. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, 26–33. AAAI.

Banihashemi, B.; Giacomo, G. D.; and Lespérance, Y. 2017. Abstraction in Situation Calculus Action Theories. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, 1048–1055. AAAI Press.

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

Cui, Z.; Liu, Y.; and Luo, K. 2021. A Uniform Abstraction Framework for Generalized Planning. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, 1837–1844. ijcai.org.

Fagin, R.; Halpern, J. Y.; Moses, Y.; and Vardi, M. Y. 1995. *Reasoning About Knowledge*. MIT Press. ISBN 9780262562003.

Giacomo, G. D.; Lespérance, Y.; and Levesque, H. J. 2000. ConGolog, a concurrent programming language based on the situation calculus. *Artif. Intell.*, 121(1-2): 109–169.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artif. Intell.*, 173(5-6): 503–535.

Kabanza, F.; Barbeau, M.; and St-Denis, R. 1997. Planning Control Rules for Reactive Agents. *Artif. Intell.*, 95(1): 67–11.

Knoblock, C. A. 1994. Automatically Generating Abstractions for Planning. *Artif. Intell.*, 68(2): 243–302.

Lin, F.; and Reiter, R. 1994. Forget It! In *Working Notes of AAAI Fall Symposium On Relevance AAAI, Menlo Park, CA, 1994*.

Lomuscio, A.; Qu, H.; and Raimondi, F. 2017. MCMAS: an open-source model checker for the verification of multi-agent systems. *Int. J. Softw. Tools Technol. Transf.*, 19(1): 9–30.

Lomuscio, A.; and Raimondi, F. 2006. The complexity of model checking concurrent programs against CTLK specifications. In *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, May 8-12, 2006*, 548–550. ACM.

Luo, K.; Liu, Y.; Lespérance, Y.; and Lin, Z. 2020. Agent Abstraction via Forgetting in the Situation Calculus. In *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020*, volume 325, 809–816. IOS Press.

McCarthy, J.; and Hayes, P. J. 1969. Some Philosophical Problems From the Standpoint of Artificial Intelligence. *Machine Intelligence*, 4: 464–502.

Nau, D. S.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN Planning System. *J. Artif. Intell. Res.*, 20: 379–404.

Raimondi, F. 2006. *Model checking multi-agent systems*. Ph.D. thesis, University College London, UK.

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

Sacerdoti, E. D. 1974. Planning in a Hierarchy of Abstraction Spaces. *Artif. Intell.*, 5(2): 115–135.

Seipp, J.; and Helmert, M. 2018. Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning. *J. Artif. Intell. Res.*, 62: 535–577.

Slaney, J. K.; and Thiébaux, S. 2001. Blocks World revisited. *Artif. Intell.*, 125(1-2): 119–153.

Vallati, M.; Chrpa, L.; Grzes, M.; McCluskey, T. L.; Roberts, M.; and Sanner, S. 2015. The 2014 International Planning Competition: Progress and Trends. *AI Mag.*, 36(3): 90–98.