A Syntactic Approach to Computing Complete and Sound Abstraction in the Situation Calculus

Liangda Fang[1, 4], Xiaoman Wang[1], Zhang Chen[1], Kailun Luo[2], Zhenhe Cui[3], Quanlong Guan[1*]

[1]Jinan University, Guangzhou 510632, China
[2]Dongguan University of Technology, Dongguan 523808, China
[3]Hunan University of Science and Technology, Xiangtan 411201, China
[4]Pazhou Lab, Guangzhou 510330, China
fangld@jnu.edu.cn, wangxm@stu2022.jnu.edu.cn, ztchen@stu2020.jnu.edu.cn,
luokl@dgut.edu.cn, zhhcui@hnust.edu.cn, gql@jnu.edu.cn

Abstract

Abstraction is an important and useful concept in the field of artificial intelligence. To the best of our knowledge, there is no syntactic method to compute a sound and complete abstraction from a given low-level basic action theory and a refinement mapping. This paper aims to address this issue. To this end, we first present a variant of situation calculus, namely linear integer situation calculus, which serves as the formalization of high-level basic action theory. We then migrate Banihashemi, De Giacomo, and Lespérance's abstraction framework to one from linear integer situation calculus to extended situation calculus. Furthermore, we identify a class of Golog programs, namely guarded actions, so as to restrict low-level Golog programs, and impose some restrictions on refinement mappings. Finally, we design a syntactic approach to computing a sound and complete abstraction from a low-level basic action theory and a restricted refinement mapping.

Extended version — http://arxiv.org/abs/2412.11217

Introduction

Abstraction plays an important role in many fields of artificial intelligence, including planning, multi-agent systems and reasoning about actions. The idea behind abstraction is to reduce a problem over large or even infinite state space to a problem over small and finite state space by aggregating similar concrete states into abstract states, and hence facilitate searching a solution for planning problems and checking some properties against multi-agent systems.

For classical planning, abstraction is often used as a heuristic function to guide the search of the solution (Helmert et al. 2014; Seipp and Helmert 2018). Generalized planning aims to find a uniform solution for possibly infinitely many problem instances (Levesque 2005; Segovia-Aguas, Jiménez, and Jonsson 2018). Srivastava, Immerman, and Zilberstein (2011) developed a method to generate a finite-state automata-based solution for generalized planning based on abstraction using 3-valued semantics (Sagiv, Reps, and Wilhelm 2002).

In multi-agent systems, abstraction is used to accelerate verification of specifications formalized in computation-tree logic (Shoham and Grumberg 2007), alternating-time temporal logic (Lomuscio and Michaliszyn 2014; Belardinelli, Ferrando, and Malvone 2023) and its epistemic extensions (Lomuscio and Michaliszyn 2016; Belardinelli, Lomuscio, and Michaliszyn 2016), strategy logic (Belardinelli et al. 2023), and μ-calculus (Ball and Kupferman 2006; Grumberg et al. 2007).

In the area of reasoning about actions, the situation calculus (Reiter 2001) is an expressive framework for modeling and reasoning about dynamic changes in the world. Golog (Levesque et al. 1997) and its concurrency extension ConGolog (De Giacomo, Lespérance, and Levesque 2000) are agent programming languages based on the situation calculus that are popular means for the control of autonomous agents. Banihashemi, De Giacomo, and Lespérance (2017) developed a general abstraction framework based on the situation calculus and ConGolog programming language. This abstraction framework relates high-level basic action theory (BAT) to low-level BAT via the notion of refinement mapping. A refinement mapping associates each high-level fluent to a low-level formula and each high-level action to a low-level ConGolog program. Based on the concept of bisimulation between high-level models and low-level models (Milner 1971), they also provide the definition of sound/complete abstractions of low-level BATs under refinement mappings.

Although a lot of efforts have made on abstractions, to the best of our knowledge, there is no syntactic approach to computing complete and sound abstractions given suitable refinement mappings. This paper aims to address this issue. To this end, we first present a variant of situation calculus, namely linear integer situation calculus, using linear integer arithmetic as the underlying logic. We then adopt linear integer situation calculus as the formalization of high-level action theory and extend Banihashemi, De Giacomo, and Lespérance (2017)'s abstraction framework. In addition, we identify a class of Golog programs, namely guarded actions, so as to restrict low-level Golog programs and impose some restrictions on refinement mappings. Finally, given a low-level BAT and a restricted refinement mapping,

*Corresponding author

we devise a syntactic approach to computing the initial knowledge base, the action precondition axioms and the successor state axioms of a complete and sound abstraction.

Preliminaries

In this section, we briefly introduce the concepts of linear integer arithmetic, the situation calculus, Golog programming language and forgetting.

Linear Integer Arithmetic

Let $\mathbb{B}$ be the set of Boolean constants $\{\top, \bot\}$ and $\mathbb{Z}$ the set of integers. The syntax of LIA-definable terms and of formulas are defined as:

$$e ::= c \mid x \mid e_1 + e_2 \mid e_1 - e_2$$
$$\phi ::= \top \mid \bot \mid p \mid e_1 = e_2 \mid e_1 < e_2 \mid e_1 \cong_c e_2 \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2$$

where $c \in \mathbb{Z}$, p is a Boolean variable and x is a numeric variable.

The formula $e_1 \cong_c e_2$ means that e_1 and e_2 are congruence modulo c, that is, $e_1 - e_2$ is divisible by c. The formula $\phi_1 \supset \phi_2$ is the shorthand for $\neg\phi_1 \vee \phi_2$, $e_1 \leq e_2$ for $e_1 = e_2 \vee e_1 < e_2$, and $e_1 \ncong_c e_2$ for $\neg(e_1 \cong_c e_2)$.

We remark that the above syntax of LIA does not involve quantifier symbols. This is because LIA admits quantifier elimination, that is, any LIA-definable formula with quantifiers can be equivalently transformed into a quantifier-free one (Cooper 1972; Monniaux 2010).

Situation Calculus

The standard situation calculus $\mathcal{L}_{sc}$ (Reiter 2001) is a first-order logic language with limited second-order features for representing and reasoning about dynamically changing worlds. There are three disjoint sorts: *action* for actions, *situation* for situations, and *object* for everything else. The constant S_0 denotes the initial situation before any action is performed; $do(a, s)$ denotes the successor situation resulting from performing action a in situation s; and a binary predicate $s \sqsubseteq s'$ means that situation s is a sub-history of situation s'. The predicate $Poss(a, s)$ describes the precondition of action a in situation s. Predicates and functions whose values may change from one situation to another are called *fluent*, taking a situation term as their last argument. For simplicity, we assume that there are no functional fluents in the standard situation calculus.

A formula ϕ with all situation arguments suppressed is called *situation-suppressed*, and $\phi[s]$ denotes the uniform formula obtained from ϕ by restoring situation variable s into all fluent names mentioned in ϕ. We say the formula $\phi[s]$ is an LIA-definable formula uniform in s, if ϕ is LIA-definable. The notion of situation-suppressed and uniform terms can be similarly defined.

A basic action theory (BAT) $\mathcal{D}$ which describes how the world changes as the result of the available actions consists of the following disjoint sets of axioms: foundational axioms Σ, initial knowledge base (KB) $\mathcal{D}_{S_0}$[1], action precondition axioms $\mathcal{D}_{ap}$, successor state axioms (SSAs) $\mathcal{D}_{ss}$, unique names axioms $\mathcal{D}_{una}$ for actions.

We introduce an abbreviation: $Exec(s) \doteq \forall a, s^*.do(a, s^*) \sqsubseteq s \supset Poss(a, s^*)$. The notation $Exec(s)$ means s is an executable situation, that is, an action history in which it is possible to perform the actions one after the other.

Extension of the Situation Calculus First-order logic is the underlying logic of the standard situation calculus. However, it lacks the mechanism for unbounded iteration (Grädel 1991) and has the limited counting ability (Kuske and Schweikardt 2017). It is necessary to represent unbounded iteration and counting ability in some domains. To represent these domains in the situation calculus, Cui, Liu, and Luo (2021) use first-order logic with transitive closure and counting as the underlying logic.

Let $\phi(\vec{x}, \vec{y})$ be a formula with two k-tuples $\vec{x}$ and $\vec{y}$ of free variables, and $\vec{u}$ and $\vec{v}$ two k-tuples of terms. $[\mathtt{TC}_{\vec{x},\vec{y}}\phi(\vec{x}, \vec{y})](\vec{u}, \vec{v})$ is the formula which means the pair $(\vec{u}, \vec{v})$ is contained in the reflexive transitive closure of the binary relation on k-tuples that is defined by ϕ. The notation $\vec{x} = \vec{y}$ abbreviates for $\bigwedge_{i=1}^{k}(x_i = y_i)$ where x_i and y_i are the i-th element of $\vec{x}$ and $\vec{y}$, respectively. Similarly, $\vec{x} \neq \vec{y}$ abbreviates for $\bigvee_i^k(x_i \neq y_i)$. For simplicity, we use $P^*(\vec{x}, \vec{y})$ for $[\mathtt{TC}_{\vec{x},\vec{y}}P(\vec{x}, \vec{y})](\vec{x}, \vec{y})$ and $P^+(\vec{x}, \vec{y})$ for $P^*(\vec{x}, \vec{y}) \wedge \vec{x} \neq \vec{y}$ where $P(\vec{x}, \vec{y})$ is a $2k$-arity predicate. Let $\phi(\vec{x})$ be a formula with a tuple $\vec{x}$ of free variables. $\#\vec{x}.\phi(\vec{x})$ is a counting term denoting the number of tuples $\vec{x}$ satisfying the formula ϕ.

To extend the situation calculus with counting, we introduce a sort *integer* for integers. If $\phi(\vec{x})$ is a formula, then $\#\vec{x}.\phi(\vec{x})$ is a term of sort integer, with the same meaning in first-order logic with transitive closure and counting. A term of sort integer is *closed*, iff every variable is bounded by the existential quantifier $\exists$ or the counting operator $\#$. In addition, we make the assumption that there are finitely many non-number objects. To do this, we adopt Li and Liu (2020)'s approach. They introduced an extra integer function symbol μ to represent a coding of objects into natural numbers. The extended basic action theory $\mathcal{D}^{\mathtt{EXT}}$ of $\mathcal{L}_{sc}^{\mathtt{EXT}}$ contains one additional axiom, namely *finitely many objects axiom*, $\mathcal{D}_{fma}$, that is, the conjunction of the following sentences:

1. $\forall x, y(\mu(x) = \mu(y) \supset x = y)$;
2. $\exists m, n \forall x(m \leq \mu(x) \wedge \mu(x) \leq n)$[2].

The first conjunction means that different objects have different codings while the second component says that there are the smallest coding and the largest one.

We explicitly provide the state constraints $\mathcal{D}_{con}$ that is a sentence of the form $\forall s.Exec(s) \supset \phi[s]$. We use the notation $\mathcal{D}_{con}^{-}$ for the situation-suppressed formula ϕ in $\mathcal{D}_{con}$.

We hereafter give an extended BAT $\mathcal{D}^{BW_l}$ for the blocks world (Slaney and Thiébaux 2001; Cook and Liu 2003).

[1]For simplicity, we assume that $\mathcal{D}_{S_0}$ is a formula uniform in S_0 rather than a set of formulas.

[2]The second conjunction in this paper is slightly different from Li and Liu (2020)'s. We introduce a sort *integer* for integers in the situation calculus while they introduced a sort for natural numbers. Hence, our axiom needs to describe the existence of the smallest coding of objects.

Example 1. *In blocks world, there is one table, one gripper and finitely many blocks. Each block is either on the table or on the other block. The gripper holds at most one block at one time. When the gripper is empty, it can pick up a block on which there is no block. The gripper can put down the block which it holds on the table or the other block. There is a special block C. Initially, the gripper is empty and at least one block is above C. The following is the formalization of blocks world. Throughout this paper, we assume that free variables are implicitly universally quantified.*

Fluents:

- $holding(x, s)$*: the gripper holds block x in situation s;*
- $on(x, y, s)$*: block x is on block y in situation s;*

Actions:

- $pickup(x)$*: pick up block x from the table;*
- $putdown(x)$*: put down block x onto the table;*
- $unstack(x, y)$*: pick up block x from block y;*
- $stack(x, y)$*: put block x onto block y;*

Initial knowledge base $\mathcal{D}_{S_0}^{BW_l}$***:***

$\neg\exists x.holding(x, S_0) \wedge \exists x.on^+(x, C, S_0) \wedge \mathcal{D}_{con}^{BW_l-}[S_0]$;

Action precondition axioms $\mathcal{D}_{ap}^{BW_l}$***:***

- $Poss(pickup(x), s) \equiv$
  $\neg(\exists y)on(y, x, s) \wedge \neg(\exists y)holding(y, s)$;
- $Poss(putdown(x), s) \equiv holding(x, s)$;
- $Poss(unstack(x, y), s) \equiv$
  $on(x, y, s) \wedge \neg(\exists z)on(z, x, s) \wedge \neg(\exists z)holding(z, s)$;
- $Poss(stack(x, y), s) \equiv$
  $holding(x, s) \wedge \neg\exists z.on(z, y, s)$;

Successor state axioms $\mathcal{D}_{ss}^{BW_l}$***:***

- $on(x, y, do(a, s)) \equiv [a = stack(x, y)] \vee$
  $[on(x, y, s) \wedge a \neq unstack(x, y)]$;
- $holding(x, do(a, s)) \equiv$
  $[a = pickup(x) \vee \exists y(a = unstack(x, y))] \vee$
  $[holding(x, s) \wedge a \neq putdown(x) \wedge$
  $\forall y(a \neq stack(x, y))]$;

State constraints $\mathcal{D}_{con}^{BW_l-}$***:*** *the conjunction of the following sentences*

- $\neg on^+(x, x)$;
- $on^+(x, y) \wedge on^+(x, z) \supset [y = z \vee on^+(y, z) \vee on^+(z, y)]$;
- $on^+(y, x) \wedge on^+(z, x) \supset [y = z \vee on^+(y, z) \vee on^+(z, y)]$;
- $holding(x) \wedge holding(y) \supset x = y$;
- $holding(x) \supset \forall y[\neg on(x, y) \wedge \neg on(y, x)]$;
- $on(x, y) \supset \neg holding(x) \wedge \neg holding(y)$. □

The first three sentences of $\mathcal{D}_{con}^{BW_l-}$ together say that the predicate $on(x, y)$ defines a collection of linear orders. The four sentence requires that $holding(x)$ is *exclusive*, that is, at most one block is holding. The last two sentences mean that the two predicates $holding(x)$ and $on(x, y)$ are *mutex*, that is, (1) if a block x is holding, then no block y is on or under block x, and (2) if a block x is on block y, then neither of them is holding.

Our state constraints are slightly different from those in (Cook and Liu 2003). First, they used a primitive predicate $above(x, y)$ to represent block x is above block y with the same meaning of the transitive closure $on^+(x, y)$ we use in this paper. The two state constraints $above(x, y) \wedge above(y, z) \supset above(x, z)$ and $above(x, y) \supset [(\exists z)on(x, z) \vee (\exists w)on(w, y)]$, presented in (Cook and Liu 2003), are redundant. Second, the finitely many objects axiom implies that every tower has a bottom and a top block. The two abbreviations $ontable \doteq \neg(\exists y)above(x, y)$ and $clear \doteq \neg(\exists y)above(y, x)$ and the two state constraints $ontable(x) \vee \exists y(above(x, y) \wedge ontable(y))$ and $clear(x) \vee \exists y(above(y, x) \wedge clear(y))$, defined in (Cook and Liu 2003), are unnecessary. Third, blocks world in this paper involves an additional predicate $holding(x)$. We need three extra state constraints for $holding(x)$.

Golog

To represent and reason about complex actions, Levesque et al. (1997) introduced a programming language, namely Golog. The syntax of Golog programs is defined as:

$$\delta ::= nil \mid a \mid \phi? \mid \delta_1; \delta_2 \mid \delta_1 | \delta_2 \mid \pi x.\delta \mid \delta^*$$

where nil is an empty program; a is an action term, possibly with variables; $\phi?$ is a test action which tests whether ϕ holds in the current situation where ϕ is a situation-suppressed formula; $\delta_1; \delta_2$ is a sequential structure of δ_1 and δ_2; $\delta_1 \mid \delta_2$ is the non-deterministic choice between δ_1 and δ_2; $\pi x.\delta$ is the program δ with some non-deterministic choice of a legal binding for variable x; δ^* means δ executes zero or more times. A Golog program is *closed*, iff every variable is bounded by the existential quantifier $\exists$, the counting operator $\#$, or the choice operator π.

The semantics of Golog programs is specified by the notation $Do(\delta, s, s')$, which means that s' is a legal terminating situation of an execution of δ starting in situation s. There are two ways to define $Do(\delta, s, s')$. Levesque et al. (1997) treated Golog programs δ as an additional extralogical symbols and hence $Do(\delta, s, s')$ is expanded into a formula by induction on the structure of δ. However, the above treatment only works for programs without concurrency and cannot qualify over programs. De Giacomo, Lespérance, and Levesque (2000) encoded ConGolog programs, the concurrency extension to Golog, as terms in the situation calculus, and define $Do(\delta, s, s')$ as an abbreviation:

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

where $Trans(\delta, s, \delta', s')$ means that a single step of program δ starting in situation s leads to situation s' with the remaining program δ' to be executed, $Trans^*$ denotes the reflexive transitive closure of $Trans$, and $Final(\delta, s)$ means that program δ may legally terminate in situation s.

The semantics of both approaches are equivalent for Golog programs (Theorem 1 in (De Giacomo, Lespérance, and Levesque 2000)). In this paper, we adopt Levesque et al. (1997)'s approach since we use only Golog programs and do not need to qualify over programs.

Forgetting

The concept of forgetting dates back to (Boole 1854), who first considered forgetting in propositional logic. Lin and Reiter (1994) studied forgetting in first-order logic.

Definition 1. Let $\mathcal{Q}$ be a set of predicate and functional symbols. We say two structures M_1 and M_2 are $\mathcal{Q}$-identical, written $M_1 \underline{\leftrightarrow}_{\mathcal{Q}} M_2$, if M_1 and M_2 agree on everything except possibly on the interpretation of every symbol of $\mathcal{Q}$.

Definition 2. Let $\mathcal{T}$ be a theory and $\mathcal{Q}$ a set of predicate and functional symbols. A theory $\mathcal{T}'$ is a result of forgetting $\mathcal{Q}$ in $\mathcal{T}$, written $\texttt{forget}(\mathcal{T}, \mathcal{Q}) \equiv \mathcal{T}'$, if for every structure M', $M' \models \mathcal{T}'$ iff there is a model M of $\mathcal{T}$ s.t. $M \underline{\leftrightarrow}_{\mathcal{Q}} M'$.

Forgetting a set of symbols results in a weak theory that has the same set of logical consequences irrelevant to the forgotten symbols as the original theory.

Proposition 1. *Let $\mathcal{T}$ be a theory and $\mathcal{Q}$ a set of predicate and functional symbols. Then, $\mathcal{T} \models \texttt{forget}(\mathcal{T}, \mathcal{Q})$, and for any sentence ϕ wherein $\mathcal{Q}$ does not appear, $\mathcal{T} \models \phi$ iff $\texttt{forget}(\mathcal{T}, \mathcal{Q}) \models \phi$.*

If two formulas ϕ and ψ are equivalent under a background theory $\mathcal{T}$, then the result of forgetting a set of symbols in the union of $\mathcal{T}$ and ψ and that in the union $\mathcal{T}$ and ψ are equivalent.

Proposition 2. *Let $\mathcal{T}$ be a theory, ϕ and ψ two formulas, and $\mathcal{Q}$ a set of predicate and functional symbols s.t. $\mathcal{T} \models \phi \equiv \psi$. Then, $\texttt{forget}(\mathcal{T} \cup \{\phi\}, \mathcal{Q}) \equiv \texttt{forget}(\mathcal{T} \cup \{\psi\}, \mathcal{Q})$.*

Linear Integer Situation Calculus

In this section, we propose a variant of the situation calculus based on LIA, namely linear integer situation calculus, that serves as a formalization of high-level BATs. The main idea is to impose a syntactic constraint of LIA on the situation calculus. In the following, we use $\mathcal{L}_{sc}^{\texttt{LIA}}$ to denote the language of linear integer situation calculus.

Linear integer situation calculus $\mathcal{L}_{sc}^{\texttt{LIA}}$ involves only three sorts integers, actions and situations and allows functional fluents. In addition, $\mathcal{L}_{sc}^{\texttt{LIA}}$ obeys the following restrictions: (1) every predicate and functional fluent contains exactly one argument of sort situation; (2) the domain of every functional fluent is the set of integers; (3) every action function has no argument, and hence is a ground action.

The linear integer basic action theory (LIBAT) $\mathcal{D}^{\texttt{LIA}} = \Sigma \cup \mathcal{D}_{S_0} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{una}$ is

- The foundational axioms for situations Σ and the set of unique names axioms for actions $\mathcal{D}_{una}$ are the same as those in the standard BAT $\mathcal{D}$.
- Initial knowledge base $\mathcal{D}_{S_0}$: a formula $\phi[S_0]$ that is an LIA-definable formula uniform in S_0.
- Action precondition axioms $\mathcal{D}_{ap}$: for each action A, there is an axiom of the form $Poss(A, s) \equiv \Pi_A[s]$, where $\Pi_A[s]$ is an LIA-definable formula uniform in s.
- Successor state axioms $\mathcal{D}_{ss}$: for each predicate fluent P, there is one axiom of the form $P(do(a, s)) \equiv \Phi_P(a, s)$ where $\Phi_P(a, s)$ is defined as:

$$[\bigvee_{i=1}^{m}(a = \alpha_i^+ \wedge \gamma_i^+[s])] \vee [P(s) \wedge \neg(\bigvee_{j=1}^{n}(a = \alpha_j^- \wedge \gamma_j^-[s]))]$$

  where each $\gamma_i^+[s]$ and $\gamma_j^-[s]$ is an LIA-definable formula uniform in s;

  and for each functional fluent f, there is one axiom of the form $f(do(a, s)) = y \equiv \Phi_f(y, a, s)$ where $\Phi_f(y, a, s)$ is defined as:

$$[\bigvee_{i=1}^{m}(y = t_i[s] \wedge a = \alpha_i^+ \wedge \gamma_i^+[s])] \vee$$
$$[y = f(s) \wedge \neg(\bigvee_{j=1}^{n}(a = \alpha_j^- \wedge \gamma_j^-[s]))]$$

  where each $\gamma_i^+[s]$ and $\gamma_j^-[s]$ is an LIA-definable formula uniform in s and each $t_i[s]$ is an LIA-definable term uniform in s.

Given an action α, $\Phi_P(\alpha, s)$ can be simplified as a formula uniform in s, denoted by $\tilde{\Phi}_{P,\alpha}[s]$, via utilizing the unique names axioms for actions to remove the comparability relation between actions. Similarly, we use $\tilde{\Phi}_{f,\alpha}(y)[s]$ to denote the formula uniform in s obtained from $\Phi_f(y, \alpha, s)$ via simplification.

We close this section by providing a LIBAT $\mathcal{D}^{BW_h}$ of blocks world.

Example 2. ***Fluents:***

- $Holding(s)$*: the gripper holds a block in situation s;*
- $Num(s)$*: the number of blocks that are above block C in situation s;*

Actions:

- $PickAboveC$*: pick up the topmost block on the tower that contains block C;*
- $Putdown$*: put down the block which the gripper holds onto the table;*

Initial knowledge base $\mathcal{D}_{S_0}^{BW_h}$:
$\neg Holding(s) \wedge Num(s) > 0$;
Action precondition axioms $\mathcal{D}_{ap}^{BW_h}$:

- $Poss(PickAboveC, s) \equiv$
  $\neg Holding(s) \wedge Num(s) > 0$;
- $Poss(Putdown, s) \equiv Holding(s)$;

Successor state axioms $\mathcal{D}_{ss}^{BW_h}$:

- $Holding(do(a, s)) \equiv [a = PickAboveC] \vee$
  $[Holding(s) \wedge a \neq Putdown]$;
- $Num(do(a, s)) = y \equiv$
  $[y = Num(s) - 1 \wedge a = PickAboveC] \vee$
  $[y = Num(s) \wedge a \neq PickAboveC]$. □

An Abstraction Framework from $\mathcal{L}_{sc}^{\texttt{LIA}}$ to $\mathcal{L}_{sc}^{\texttt{EXT}}$

In this section, based on Banihashemi, De Giacomo, and Lespérance (2017)'s work, we develop an abstraction framework from linear integer situation calculus to extended situation calculus.

We assume that we are given two BATs $\mathcal{D}^l$ and $\mathcal{D}^h$. We consider $\mathcal{D}^l$ formulated in $\mathcal{L}_{sc}^{\texttt{EXT}}$ as the low-level BAT and $\mathcal{D}^h$ formulated in $\mathcal{L}_{sc}^{\texttt{LIA}}$ as the high-level BAT. The low-level BAT $\mathcal{D}^l$ contains a finite set of action types $\mathcal{A}^l$ and a finite set of predicate fluents $\mathcal{P}^l$. Given a low-level situation-suppressed formula ϕ, we use $\mathcal{P}^l(\phi)$ for the set of predicate fluents that occur in ϕ. The meaning of the two notations $\mathcal{A}^h$ and $\mathcal{P}^h$ are similar for the high-level BAT

$\mathcal{D}^h$, and $\mathcal{F}^h$ stands for a finite set of high-level functional fluents. Given a high-level situation-suppressed formula ϕ, we use $\mathcal{P}^h(\phi)$ (resp. $\mathcal{F}^h(\phi)$) for the set of predicate (resp. functional) fluents that occur in ϕ.

The concept of refinement mapping from $\mathcal{D}^h$ to $\mathcal{D}^l$ is the same as that proposed by Banihashemi, De Giacomo, and Lespérance (2017) except the following. (1) We allow functional fluents in high-level BATs. (2) The refinement mapping assigns every high-level action to a closed low-level Golog program since every high-level action has no argument. (3) The refinement mapping assigns every high-level predicate fluent to a closed low-level formula since every high-level predicate fluent contains exactly one argument of sort situation. (4) The refinement mapping assigns every high-level functional fluent to a closed counting term since the image of every high-level functional fluent is the set of integers.

Definition 3. We say that a function m is a refinement mapping from $\mathcal{D}^h$ to $\mathcal{D}^l$, iff

1. for every high-level action α, $m(\alpha) \doteq \delta_\alpha$ where δ_α is a closed Golog program defined over $\mathcal{A}^l$ and $\mathcal{P}^l$;
2. for every high-level predicate fluent P, $m(P) \doteq \phi_P$ where ϕ_P is a closed situation-suppressed formula defined over $\mathcal{P}^l$;
3. for every high-level functional fluent f, $m(f) \doteq \#\vec{x}.\phi_f(\vec{x})$ where ϕ_f is a situation-suppressed formula with a tuple $\vec{x}$ of free variables defined over $\mathcal{P}^l$.

Given a refinement mapping m and a high-level situation-suppressed formula ϕ, $m(\phi)$ denotes the result of replacing every occurrence of high-level predicate fluent P and functional fluent f by the low-level counterpart $m(P)$ and $m(f)$, respectively. The mapping formula Ψ_m based on m, is defined as $[\bigwedge_{P\in\mathcal{P}^h} P \equiv m(P)] \wedge [\bigwedge_{f\in\mathcal{F}^h} f = m(f)]$. Intuitively, Ψ_m says that every high-level predicate fluent has the same Boolean value as its corresponding low-level closed formula defined by m, and similarly for every high-level functional fluent.

Example 3. *A refinement mapping m from $\mathcal{D}^{BW_h}$ to $\mathcal{D}^{BW_l}$ is defined as:*

- $m(Num) \doteq \#x.on^+(x, C)$;
- $m(Holding) \doteq \exists x.holding(x)$;
- $m(PickAboveC) \doteq (\pi x, y)on^+(x, C)?; unstack(x, y)$;
- $m(Putdown) \doteq \pi x.putdown(x)$. □

We hereafter introduce the concept of m-isomorphism between high-level situations and low-level situations. We assume that M^h and M^l are models of $\mathcal{D}^h$ and $\mathcal{D}^l$, respectively. The two assignments $v[s/s_h]$ and $v[s/s_l]$ are the same as v except mapping variable s to situation s_h and s_l, respectively.

Definition 4. Given a refinement mapping m, we say that situation s_h in M^h is m-isomorphic to situation s_l in M^l, written $s_h \sim_m^{M^h,M^l} s_l$, iff

- for every high-level predicate fluent P and every variable assignment v, $M^h, v[s/s_h] \models P(s)$ iff $M^l, v[s/s_l] \models m(P)[s]$.
- for every high-level functional fluent f and every variable assignment v, $M^h, v[s/s_h] \models f(s) = y$ iff $M^l, v[s/s_l] \models m(f)[s] = y$.

We can get the following results:

Proposition 3. *If $s_h \sim_m^{M^h,M^l} s_l$, then for every high-level situation-suppressed formula ϕ and every variable assignment v, $M^h, v[s/s_h] \models \phi[s]$ iff $M^l, v[s/s_l] \models m(\phi)[s]$.*

Proposition 4. *If $s_h \sim_m^{M^h,M^l} s_l$, then for every low-level situation-suppressed formula ϕ and every variable assignment v, we have $M^l, v[s/s_l] \models \phi[s] \wedge \mathcal{D}^l_{fma}$ implies that $M^h, v[s/s_h] \models \texttt{forget}(\phi \wedge \mathcal{D}^l_{fma} \wedge \Psi_m, \mathcal{P}^l \cup \{\mu\})[s]$.*

To relate high-level and low-level theories, we resort to m-bisimulation relation between high-level models and low-level models. The notation Δ_S^M stands for the situation domain of M.

Definition 5. A relation $B \subseteq \Delta_S^{M^h} \times \Delta_S^{M^l}$ is an m-bisimulation between M^h and M^l, iff $(s_h, s_l) \in B$ implies that

atom s_h and s_l are m-isomorphic.

forth for every high-level action α and every low-level situation s'_l s.t. $M^l, v[s/s_l, s'/s'_l] \models Do(m(\alpha), s, s')$, there is a high-level situation s'_h s.t. $M^h, v[s/s_h, s'/s'_h] \models Poss(\alpha, s) \wedge s' = do(\alpha, s)$ and $(s'_h, s'_l) \in B$.

back for every high-level action α and every high-level situation s'_h s.t. $M^h, v[s/s_h, s'/s'_h] \models Poss(\alpha, s) \wedge s' = do(\alpha, s)$, there is a low-level situation s'_l s.t. $M^l, v[s/s_l, s'/s'_l] \models Do(m(\alpha), s, s')$ and $(s'_h, s'_l) \in B$.

We say M^h is m-bisimilar to M^l, written $M^h \sim_m M^l$, iff there is an m-bisimulation B between M^h and M^l s.t. $(S_0^{M^h}, S_0^{M^l}) \in B$.

Based on the notion of m-bisimulation, we hereafter provide definitions of sound and complete abstraction of low-level theories.

Definition 6. We say $\mathcal{D}^h$ is a *sound abstraction* of $\mathcal{D}^l$ relative to a refinement mapping m, iff for every model M^l of $\mathcal{D}^l$, there is a model M^h of $\mathcal{D}^h$ s.t. $M^h \sim_m M^l$.

Let m be a refinement mapping. Let π_m be a low-level program $[m(\alpha_1)|\cdots|m(\alpha_n)]$ where each $\alpha_i \in \mathcal{A}^h$. Intuitively, π_m denotes any refinement of any high-level primitive actions. The iteration π_m^* denotes any refinement of any high-level action sequence.

Theorem 1. *$\mathcal{D}^h$ is a sound abstraction of $\mathcal{D}^l$ relative to a refinement mapping m iff the following hold*

1. $\mathcal{D}^l_{S_0} \wedge \mathcal{D}^l_{fma} \models m(\phi)[S_0]$ *where* $\mathcal{D}^h_{S_0} = \phi[S_0]$;
2. $\mathcal{D}^l \models \forall s.Do(\pi_m^*, S_0, s) \supset \bigwedge_{\alpha\in\mathcal{A}^h}(m(\Pi_\alpha)[s] \equiv \exists s' Do(m(\alpha), s, s'))$;
3. $\mathcal{D}^l \models \forall s.Do(\pi_m^*, S_0, s) \supset \bigwedge_{\alpha\in\mathcal{A}^h} \forall s'\{Do(m(\alpha), s, s') \supset \bigwedge_{P\in\mathcal{P}^h}(m(P)[s'] \equiv m(\tilde{\Phi}_{P,\alpha})[s]) \wedge \bigwedge_{f\in\mathcal{F}^h} \forall y(m(f)[s'] = y \equiv m(\tilde{\Phi}_{f,\alpha})(y)[s])\}$.

Definition 7. We say $\mathcal{D}^h$ is a *complete abstraction* of $\mathcal{D}^l$ relative to a refinement mapping m, iff for every model M^h of $\mathcal{D}^h$, there is a model M^l of $\mathcal{D}^l$ s.t. $M^h \sim_m M^l$.

If the high-level BAT $\mathcal{D}^h$ is a sound abstraction of the low-level BAT $\mathcal{D}^l$, then deciding if $\mathcal{D}^h$ is a complete abstraction of $\mathcal{D}^l$ reduces to determining if the high-level initial knowledge base $\mathcal{D}^h_{S_0}$ is the result of forgetting all low-level predicate fluents and the coding symbol in the union of the low-level initial knowledge base, the finitely many objects axiom and the mapping formula.

Theorem 2. *Suppose that $\mathcal{D}^h$ is a sound abstraction of $\mathcal{D}^l$ relative to a refinement mapping m. Then, $\mathcal{D}^h$ is a complete abstraction of $\mathcal{D}^l$ relative to m iff $\mathcal{D}^h_{S_0} \equiv \texttt{forget}(\phi \wedge \mathcal{D}^l_{fma} \wedge \Psi_m, \mathcal{P}^l \cup \{\mu\})[S_0]$ where $\phi[S_0] = \mathcal{D}^l_{S_0}$.*

Computing Sound and Complete Abstraction

In this section, we first identify a class of Golog programs, namely *guarded actions*, and connect high-level actions to low-level guarded actions. We then impose some restrictions on refinement mappings and provide a syntactic approach to computing the initial KB, action precondition axioms and successor state axioms of sound and complete abstraction from a low-level BAT and a restricted refinement mapping.

Guarded Actions

We say a Golog program δ is a *guarded action*, if it is of the form $\pi\vec{x}.\psi(\vec{x})?; A(\vec{x},\vec{c})$. A guarded action requires all variables $\vec{x}$ of action $A(\vec{x},\vec{c})$ to be guarded by the formula $\psi(\vec{x})$.

Definition 8. Let $A(\vec{x},\vec{c})$ be a low-level action with a tuple $\vec{x}$ of variables and a tuple $\vec{c}$ of constants. Let $\phi(\vec{y})$ and $\psi(\vec{x})$ be two low-level situation-suppressed formulas where $\vec{x}$ includes all of the variables of $\vec{y}$. We say $A(\vec{x},\vec{c})$ is

- *alternately $\phi(\vec{y})$-enabling* under $\psi(\vec{x})$, iff $\mathcal{D}^l \models \forall s,\vec{x}.(Poss(A(\vec{x},\vec{c}),s) \wedge \psi(\vec{x})[s]) \supset (\neg\phi(\vec{y})[s] \wedge \phi(\vec{y})[do(A(\vec{x},\vec{c}),s)])$;
- *singly $\phi(\vec{y})$-enabling* under $\psi(\vec{x})$, iff $A(\vec{x},\vec{c})$ is alternately $\phi(\vec{y})$-*enabling* under $\psi(\vec{x})$ and $\mathcal{D}^l \models \forall s,\vec{x}.(Poss(A(\vec{x},\vec{c}),s) \wedge \psi(\vec{x})[s]) \supset \forall\vec{z}(\vec{y} \neq \vec{z} \supset \phi(\vec{z})[s] \equiv \phi(\vec{z})[do(A(\vec{x},\vec{c}),s)])$.

Definition 9. Let $A(\vec{x},\vec{c})$ be a low-level action with a tuple $\vec{x}$ of variables and a tuple $\vec{c}$ of constants. Let $\phi(\vec{y})$ and $\psi(\vec{x})$ be two low-level situation-suppressed formulas. We say $A(\vec{x},\vec{c})$ is $\phi(\vec{y})$-*invariant* under $\psi(\vec{x})$, iff $\mathcal{D}^l \models \forall s,\vec{x}.(Poss(A(\vec{x},\vec{c}),s) \wedge \psi(\vec{x})[s]) \supset \forall\vec{y}(\phi(\vec{y})[s] \equiv \phi(\vec{y})[do(A(\vec{x},\vec{c}),s)])$.

Alternative enabling property says that (1) $\phi(\vec{y})$ does not hold in the situation s in which both the precondition of action $A(\vec{x},\vec{c})$ and the condition $\psi(\vec{x})$ holds, and (2) $\phi(\vec{y})$ holds after performing $A(\vec{x},\vec{c})$. Single enabling property is a stronger property than alternative enabling, stipulating that $A(\vec{x},\vec{c})$ does not affect the Boolean value of $\phi(\vec{z})$ for any tuple $\vec{z}$ of objects not equal to $\vec{y}$. Invariance property means that $A(\vec{x},\vec{c})$ does not affect the truth value of $\phi(\vec{y})$ with any tuple $\vec{y}$.

Definition 10. Let $\phi(\vec{x})$ be a low-level situation suppressed formula. We say $\phi(\vec{x})$ is *exclusive*, iff $\mathcal{D}^l \models \forall s,\vec{x},\vec{y}.Exec(s) \supset (\phi(\vec{x})[s] \wedge \phi(\vec{y})[s] \supset \vec{x} = \vec{y})$.

Intuitively, at most one tuple $\vec{x}$ satisfies the formula $\phi(\vec{x})$ in every executable situation.

Example 4. *We continue with the low-level BAT $\mathcal{D}^{BW_l}$ of blocks world illustrated in Example 1. For any situation s, the action precondition axiom for $putdown(x)$ entails $holding(x,s)$ and the successor state axioms entails $\neg holding(x, do(putdown(x),s))$. Therefore, $putdown(x)$ is alternately $\neg holding(x)$-enabling under $\top$. In addition, performing action $putdown(x)$ does not change the Boolean value of $on^+(y,C)$ of any object y and hence $putdown(x)$ is $on^+(y,C)$-invariant under $\top$.*

Similarly, $unstack(x,y)$ is alternately $on^+(x,C)$-enabling under $on^+(x,C)$ and alternately $holding(x)$-enabling under $\top$. Furthermore, when $on^+(x,C)$ holds, performing $unstack(x,y)$ only grasps the topmost block on the tower contains block C and does not catch other blocks on the same tower. Hence, $unstack(x,y)$ is singly $on^+(x,C)$-enabling under $on^+(x,C)$.

The fourth state constraint $holding(x) \wedge holding(y) \supset x = y$ deduces that $holding(x)$ is exclusive. □

We hereafter define connections between low-level guarded actions together with the low-level situation-suppressed formulas and its corresponding high-level actions together with high-level predicate (or functional) fluents.

Definition 11. Let m be a refinement mapping, α a high-level action, P a high-level predicate fluent and f a high-level functional fluent. We say

1. α is P-enabling relative to m, iff $m(\alpha) \doteq \pi\vec{x}.\psi(\vec{x})?; A(\vec{x},\vec{c})$, $m(P) \doteq \exists\vec{y}.\phi(\vec{y})$ and $A(\vec{x},\vec{c})$ is alternately $\phi(\vec{y})$-enabling under $\psi(\vec{x})$;
2. α is P-disabling relative to m, iff $m(\alpha) \doteq \pi\vec{x}.\psi(\vec{x})?; A(\vec{x},\vec{c})$, $m(P) \doteq \exists\vec{y}.\phi(\vec{y})$, $A(\vec{x},\vec{c})$ is alternately $\neg\phi(\vec{y})$-enabling under $\psi(\vec{x})$ and $\phi(\vec{y})$ is exclusive;
3. α is P-invariant relative to m, iff $m(\alpha) \doteq \pi\vec{x}.\psi(\vec{x})?; A(\vec{x},\vec{c})$, $m(P) \doteq \exists\vec{y}.\phi(\vec{y})$ and $A(\vec{x},\vec{c})$ is $\phi(\vec{y})$-invariant under $\psi(\vec{x})$;
4. α is f-incremental relative to m, iff $m(\alpha) \doteq \pi\vec{x}.\psi(\vec{x})?; A(\vec{x},\vec{c})$, $m(f) \doteq \#\vec{y}.\phi(\vec{y})$ and $A(\vec{x},\vec{c})$ is singly $\phi(\vec{y})$-enabling under $\psi(\vec{x})$;
5. α is f-decremental relative to m, iff $m(\alpha) \doteq \pi\vec{x}.\psi(\vec{x})?; A(\vec{x},\vec{c})$, $m(f) \doteq \#\vec{y}.\phi(\vec{y})$ and $A(\vec{x},\vec{c})$ is singly $\neg\phi(\vec{y})$-enabling under $\psi(\vec{x})$;
6. α is f-invariant relative to m, iff $m(\alpha) \doteq \pi\vec{x}.\psi(\vec{x})?; A(\vec{x},\vec{c})$, $m(f) \doteq \#\vec{y}.\phi(\vec{y})$ and $A(\vec{x},\vec{c})$ is $\phi(\vec{y})$-invariant under $\psi(\vec{x})$.

The following proposition provides the effect of executing high-level action α with properties illustrated in Definition 11 when the high-level BAT is a complete abstraction of the low-level BAT.

Proposition 5. *Suppose that $\mathcal{D}^h$ is a complete abstraction of $\mathcal{D}^l$ relative to m. Let α be a high-level action. Then, the following holds:*

1. *If α is P-enabling relative to m, then $\mathcal{D}^h \models Exec(s) \wedge Poss(\alpha,s) \supset P(do(\alpha,s))$;*
2. *If α is P-disabling relative to m, then $\mathcal{D}^h \models Exec(s) \wedge Poss(\alpha,s) \supset \neg P(do(\alpha,s))$;*
3. *If α is P-invariant relative to m, then $\mathcal{D}^h \models Exec(s) \wedge Poss(\alpha,s) \supset P(do(\alpha,s)) \equiv P(s)$;*

4. *If α is f-incremental relative to m, then $\mathcal{D}^h \models Exec(s) \wedge Poss(\alpha, s) \supset f(do(\alpha, s)) = f(s) + 1$;*
5. *If α is f-decremental relative to m, then $\mathcal{D}^h \models Exec(s) \wedge Poss(\alpha, s) \supset f(do(\alpha, s)) = f(s) - 1$;*
6. *If α is f-invariant relative to m, then $\mathcal{D}^h \models Exec(s) \wedge Poss(\alpha, s) \supset f(do(\alpha, s)) = f(s)$.*

Example 5. *Example 2 presents high-level BAT $\mathcal{D}^{BW_h}$ of blocks world, which will be later verified as a sound and complete abstraction of $\mathcal{D}^{BW_l}$. Example 3 illustrates a refinement mapping m where $m(Holding) \doteq \exists x.holding(x)$ and $m(Putdown) \doteq \pi x.putdown(x)$. Example 4 shows that $putdown(x)$ is alternately $\neg holding(x)$-enabling under $\top$ and $holding(x)$ is exclusive. Hence, high-level action $Putdown$ is $Holding$-disabling relative to m. By Proposition 5, $Holding$ does not hold by performing action $Putdown$ in every high-level executable situation.*

Similarly, $Putdown$ is Num-invariant relative to m while $PickAboveC$ is $Holding$-enabling and Num-decremental. By Proposition 5, $Holding$ becomes true and the value of Num decrements by 1 via performing action $PickAboveC$ in every high-level executable situation. □

Restrictions on Refinement Mapping

We say a refinement mapping m is *flat*, iff the following two conditions hold: (1) for every high-level action α, $m(\alpha) \doteq \pi\vec{x}.\psi(\vec{x})?; A(\vec{x}, \vec{c})$; and (2) for every high-level predicate fluent P, $m(P) \doteq \exists\vec{x}.\phi(\vec{x})$. Given a flat refinement mapping m, we use $\Phi(m)$ for the set $\{\phi(\vec{x}) \mid m(P) \doteq \exists\vec{x}.\phi(\vec{x})$ for a high-level predicate fluent P or $m(f) \doteq \#\vec{x}.\phi(\vec{x})$ for a high-level functional fluent $f\}$.

We say a flat refinement mapping m is *complete*, iff the following two conditions hold: (1) for every high-level action α and every high-level predicate fluent P, α is P-enabling, P-disabling, or P-invariant; and (2) for every high-level action α and every high-level functional fluent f, α is f-incremental, f-decremental, or f-invariant.

We say a low-level situation s_l is *m-reachable*, iff $M^l, v[s/s_l] \models Do(\pi^*_m, S_0, s)$. We say two low-level situations s_1 and s_2 of two low-level structures M_1 and M_2 are *in the same abstract state*, iff the following two conditions hold: (1) for every high-level predicate fluent P and every variable assignment v, we have $M_1, v[s/s_1] \models m(P)[s]$ iff $M_2, v[s/s_2] \models m(P)[s]$; and (2) for every high-level predicate fluent f and every variable assignment v, $M_1, v[s/s_1] \models m(f)[s] = y$ iff $M_2, v[s/s_2] \models m(f)[s] = y$.

We say a refinement mapping m is *executability-preserving*, iff for every two low-level m-reachable situations s_1 and s_2 in the same abstract state, we have $M_1, v[s/s_1] \models \exists s'.Do(m(\alpha), s, s')$ iff $M_2, v[s/s_2] \models \exists s'.Do(m(\alpha), s, s')$ for every high-level action α.

Theorem 3. *Suppose that the refinement mapping m is flat, complete and executability-preserving. Let $\mathcal{D}^h = \Sigma \cup \mathcal{D}^h_{S_0} \cup \mathcal{D}^h_{ap} \cup \mathcal{D}^h_{ss} \cup \mathcal{D}^h_{una}$ where*

- *$\mathcal{D}^h_{S_0} \equiv \mathtt{forget}(\phi \wedge \mathcal{D}^l_{fma} \wedge \Psi_m, \mathcal{P}^l \cup \{\mu\})[S_0]$ where $\phi[S_0] \equiv \mathcal{D}^l_{S_0}$;*
- *$\mathcal{D}^h_{ap}$ contains the set of sentences: for every high-level action α, $Poss(\alpha, s) \equiv \mathtt{forget}((\exists\vec{x}.\psi(\vec{x}) \wedge \Pi_A(\vec{x}, \vec{c})) \wedge \mathcal{D}^{l-}_{con} \wedge \mathcal{D}^l_{fma} \wedge \Psi_m, \mathcal{P}^l \cup \{\mu\})[s]$ where $m(\alpha) \doteq \pi\vec{x}.\psi(\vec{x})?; A(\vec{x}, \vec{c})$;*
- *$\mathcal{D}^h_{ss}$ contains the set of sentences:*
  - *for every high-level predicate fluent P,*

$$P(do(a, s)) \equiv [\bigvee_{i=1}^{m} a = \alpha_i^+] \vee [P(s) \wedge \bigwedge_{j=1}^{n} a \neq \alpha_j^-]$$

  where each α_i^+ is P-enabling relative to m and each α_j^- is P-disabling.
  - *for every high-level functional fluent f,*

$$\begin{aligned} f(do(a, s)) = y \equiv [y = f(s) + 1 \wedge (\bigvee_{i=1}^{m} a = \alpha_i^+)] \vee \\ [y = f(s) - 1 \wedge (\bigvee_{i=1}^{n} a = \alpha_i^-)] \vee \\ [y = f(s) \wedge \bigwedge_{i=1}^{m} a \neq \alpha_i^+ \wedge \bigwedge_{i=1}^{n} a \neq \alpha_i^-] \end{aligned}$$

  where each α_i^+ is f-incremental relative to m and each α_i^- is f-decremental.

Then, $\mathcal{D}^h$ is a complete and sound abstraction of $\mathcal{D}^l$ relative to m.

By Theorem 3, we can directly acquire the successor state axioms of a complete and sound abstraction. The high-level initial KB can be computed via forgetting every low-level predicate fluent symbol and the coding symbol in the low-level initial KB conjoining with the finitely many objects axiom and the mapping formula. The action precondition axioms can be similarly obtained.

Example 6. *The high-level BAT $\mathcal{D}^{BW_h}$ contains a predicate fluent $Holding$, a functional fluent Num and two actions: $Putdown$ and $PickAboveC$. As mentioned in Example 5, high-level action $Putdown$ is $Holding$-disabling and Num-invariant relative to m while $PickAboveC$ is $Holding$-enabling and Num-decremental. Hence, m is flat and complete. In addition, m is also executability-preserving, which will be verified later.*

By Theorem 3, we construct the successor state axiom of the complete and sound abstraction $\mathcal{D}^{BW_h}$ of $\mathcal{D}^{BW_l}$.

Since action $Putdown$ is $Holding$-disabling relative to m and $PickAboveC$ is $Holding$-enabling, the successor state axiom for predicate fluent $Holding$ is

$$\begin{aligned} &Holding(do(a, s)) \equiv \\ &\quad [a = PickAboveC] \vee [Holding(s) \wedge a \neq Putdown]. \end{aligned}$$

As $PickAboveC$ is Num-decremental relative to m, the successor state axiom for functional fluent Num is

$$\begin{aligned} &Num(do(a, s)) = y \equiv \\ &\quad [y = Num(s) - 1 \wedge a = PickAboveC] \vee \\ &\quad [y = Num(s) \wedge a \neq PickAboveC]. \end{aligned}$$

□

In the following, we define a normal form of low-level situation suppressed formulas ϕ, namely *propositional $\exists$-formula*, and a translation m^{-1} that maps every low-level propositional $\exists$-formula to a high-level formula. We also identify four conditions on flat refinement mapping under

the translation m^{-1} derives the result of forgetting for computing the initial KB and the action precondition axioms.

Let Φ be a set of formulas. We say a formula is a *propositional $\exists$-formula over Φ*, iff it is the Boolean combination of $\exists\vec{x}.\phi(\vec{x})$'s where $\phi(\vec{x}) \in \Phi$.

We say a flat refinement mapping m is *simply forgettable*, iff $\texttt{forget}(\phi \wedge \mathcal{D}^{l-}_{con} \wedge \mathcal{D}^{l}_{fma} \wedge \Psi_m, \mathcal{P}^l \cup \{\mu\}) \equiv \texttt{forget}(\phi \wedge \Psi_m, \mathcal{P}^l)$ for every propositional $\exists$-formula ϕ over $\Phi(m)$. Simple forgettability property says that forgetting all low-level predicate fluent symbols and the coding symbol in any propositional $\exists$-formula ϕ together with the state constraint $\mathcal{D}^{l-}_{con}$, the finitely many object axiom $\mathcal{D}^{l}_{fma}$ and the mapping formula Ψ_m can be simplified to forgetting all low-level predicate fluent symbols in the conjunction of ϕ and Ψ_m.

Proposition 6. *The refinement mapping m for blocks world defined in Example 3 is simply forgettable.*

Given a flat refinement mapping m and a low-level propositional $\exists$-formula φ over $\Phi(m)$, the formula $m^{-1}(\varphi)$ is obtained from φ by the following two steps:

1. generate the formula ψ by replacing every occurrence of $\exists\vec{x}.\phi(\vec{x})$ in φ by P (resp. $f > 0$) where $m(P) \doteq \exists\vec{x}.\phi(\vec{x})$ (resp. $m(f) \doteq \#\vec{x}.\phi(\vec{x})$);
2. generate the formula $m^{-1}(\varphi)$ by conjoining ψ with the conjunct $\bigwedge_{f \in \mathcal{F}^h(\psi)} f \geq 0$.

We say a flat refinement mapping m is *consistent*, iff $\exists\vec{x}.\phi(\vec{x})$ is consistent for every high-level predicate fluent P and every high-level functional fluent f s.t. $m(P) \doteq \exists\vec{x}.\phi(\vec{x})$ and $m(f) \doteq \#\vec{x}.\phi(\vec{x})$.

We say a flat refinement mapping m is *syntax-irrelevant*, iff for every two high-level predicate (or functional) fluents P_1 (or f_1) and P_2 (or f_2) s.t. $m(P_1) \doteq \exists\vec{x}.\phi_1(\vec{x})$ (or $m(f_1) \doteq \#\vec{x}.\phi_1(\vec{x})$) and $m(P_2) \doteq \exists\vec{y}.\phi_2(\vec{y})$ (or $m(f_2) \doteq \#\vec{y}.\phi_2(\vec{y})$), $\mathcal{P}^l(\phi_1) \cap \mathcal{P}^l(\phi_2) = \emptyset$. Syntax-irrelevant refinement mapping requires that for every two high-level predicate (or functional) fluents, the two sets of low-level predicate fluents that occur in the corresponding two low-level situation-suppressed formulas ϕ_1 and ϕ_2 defined in m are disjoint.

Proposition 7. *Let m be a flat, consistent and syntax-irrelevant refinement mapping and ϕ a propositional $\exists$-formula over $\Phi(m)$. Then, $m^{-1}(\phi) \equiv \texttt{forget}(\phi \wedge \Psi_m, \mathcal{P}^l)$.*

Proposition 7 says that the result of forgetting in a propositional $\exists$-formula conjoining with the mapping formula can be obtained by the translation m^{-1}. However, the low-level initial KB and the executability condition of low-level guarded actions may not be in the form of propositional $\exists$-formulas. We therefore impose one additional condition on refinement mappings such that the above two formulas are equivalent to propositional $\exists$-formulas under the finitely many objects axiom and the state constraints, respectively.

We say a flat refinement mapping m is *propositional $\exists$-definable*, iff (1) there is a situation-suppressed propositional $\exists$-formula ϕ over $\Phi(m)$ s.t. $\mathcal{D}^{l-}_{con} \wedge \mathcal{D}^{l}_{fma} \models \phi \equiv \varphi$ where $\varphi[S_0] = \mathcal{D}^{l}_{S_0}$; and (2) for every high-level action α s.t. $m(\alpha) \doteq \pi\vec{x}.\psi(\vec{x})?; A(\vec{x},\vec{c})$, there is a situation-suppressed propositional $\exists$-formula ϕ over $\Phi(m)$ s.t. $\mathcal{D}^{l-}_{con} \wedge \mathcal{D}^{l}_{fma} \models \phi \equiv \exists\vec{x}.\psi(\vec{x}) \wedge \Pi_A(\vec{x},\vec{c})$.

Proposition 8. *A flat, consistent, syntax-irrelevant and propositional $\exists$-definable refinement mapping is executability-preserving.*

Proposition 8 means that the four properties of refinement mappings: flatness, consistency, syntax-irrelevance and propositional $\exists$-definability together guarantees the executability-preserving property.

By putting Theorem 3 and Propositions 2, 7 and 8 into together, we obtain a syntactic approach to computing sound and complete abstractions given a flat, complete, simply forgettable, consistent, syntax-irrelevant and propositional $\exists$-definable refinement mapping. We illustrate this approach with the following example.

Example 7. *Remember that $m(Num) \doteq \#x.on^+(x,C)$ and $m(Holding) \doteq \exists x.holding(x)$. The two formulas $\exists x.on^+(x,C)$ and $\exists x.holding(x)$ are consistent, so m is consistent. In addition, on and $holding$ are different low-level predicate fluent symbols, hence m is syntax-irrelevant. The set $\Phi(m)$ of formulas is $\{on^+(x,C), holding(x)\}$.*

Example 2 presents the sound and complete abstraction $\mathcal{D}^{BW_h}$ of $\mathcal{D}^{BW_l}$. We will show the construction of $\mathcal{D}^{BW_h}$ according to Theorem 3 and Propositions 2, 6 - 8. The successor state axioms was shown in Example 6.

We first construct the high-level initial KB $\mathcal{D}^{BW_h}_{S_0}$. The low-level initial KB $\mathcal{D}^{BW_l}_{S_0}$ is $(\varphi \wedge \mathcal{D}^{BW_l}_{con})[S_0]$ where $\varphi : \neg\exists x.holding(x) \wedge \exists x.on^+(x,C)$ is a propositional $\exists$-formula over $\Phi(m)$. Clearly, $\mathcal{D}^{BW_l-}_{con} \wedge \mathcal{D}^{BW_l}_{fma} \models \varphi \equiv (\varphi \wedge \mathcal{D}^{BW_l}_{con})$. The formula $m^{-1}(\varphi) = \neg Holding \wedge Num > 0 \wedge Num \geq 0$. Hence, we get that $\mathcal{D}^{BW_h}_{S_0} \equiv \neg Holding(S_0) \wedge Num(S_0) > 0$.

We then construct the action precondition axioms $\mathcal{D}^{BW_h}_{ap}$. For high-level action $Putdown$, $m(Putdown) \doteq \pi x.putdown(x)$ and $Poss(putdown(x), s) \equiv holding(x,s)$. The formula $\exists x.holding(x)$ is a propositional $\exists$-formula over $\Phi(m)$. Since $m^{-1}(\exists x.holding(x)) = Holding$, we obtain that $Poss(Putdown, s) \equiv Holding(s)$.

For high-level action $PickAboveC$, $m(PickAboveC) \doteq (\pi x, y)on^+(x,C)?; unstack(x,y)$ and $Poss(unstack(x,y), s) \equiv on(x,y,s) \wedge \neg(\exists z)on(z,x,s) \wedge \neg(\exists z)holding(z,s)$. The executability condition ϕ of guaraded action $(\pi x, y)on^+(x,C)?; unstack(x,y)$ is $\exists x, y.on^+(x,C) \wedge on(x,y) \wedge \neg(\exists z)on(z,x) \wedge \neg(\exists z)holding(z)$. Clearly, ϕ is not a propositional $\exists$-formula over $\Phi(m)$. By the transitive closure property of $on^+(x,y)$, the finitely many objects axiom and the first state constraint $\neg on^+(x,x)$ of $\mathcal{D}^{BW_l-}_{con}$, we get that $\mathcal{D}^{BW_l-}_{con} \wedge \mathcal{D}^{BW_l}_{fma} \models (\exists x)on^+(x,C) \equiv (\exists x, y.on^+(x,C) \wedge on(x,y) \wedge \neg(\exists z)on(z,x))$. It follows that $\mathcal{D}^{BW_l-}_{con} \wedge \mathcal{D}^{BW_l}_{fma} \models [(\exists x)on^+(x,C) \wedge \neg(\exists x)holding(x)] \equiv \phi$. In addition, $m^{-1}((\exists x)on^+(x,C) \wedge \neg(\exists x)holding(x)) = Num > 0 \wedge \neg Holding \wedge Num \geq 0$. Hence, $Poss(PickAboveC, s) \equiv Holding(s) \wedge Num(s) > 0$. □

Related Work

In generalized planning, Bonet, Francès, and Geffner (2019) utilize SAT solvers and description logics to learn an abstract qualitative numerical problem (QNP) (Srivastava et al. 2011) from a finite subset of problem instances. However, the learned QNP may not be a sound abstraction for the whole set of original problem instances. Later, Bonet et al. (2019) studied the necessary condition of problem instances on which the abstraction is guaranteed to be sound. In addition, QNP under deterministic semantics (that is, each integer variable x can be decremented by 1 if $x > 0$ or incremented by 1) (Srivastava et al. 2015) and generalized linear integer numeric planning problem (Lin et al. 2022) can be formalized in linear integer situation calculus.

The abstraction-refinement framework is a popular paradigm for verifying properties against multi-agent systems. It first initializes an abstract model based on 3-valued semantics. If the property is true (or false) in the abstract model, then it is also true (or false) in the concrete model. Otherwise, the abstract model is too coarse to decide the satisfaction of the property. In this case, the framework will search some failure states to refine the abstraction in an attempt to return a definite answer. Various abstraction and refinement strategies are proposed. (1) Belardinelli, Ferrando, and Malvone (2023) constructs an initial abstract model based on the common knowledge relationships among agents, and then refines the abstract model by splitting the failure state s according to the incoming transitions into s. (2) Predicate abstraction (Lomuscio and Michaliszyn 2016; Belardinelli, Lomuscio, and Michaliszyn 2016) first selects key predicates to represent the abstract model and then updates the predicate set according to failure states so as to refine the model. (3) Lomuscio and Michaliszyn (2014) combines state abstraction and action abstraction to constructs an abstract model, and then refines the model by splitting those states where the truth value of some subformula of the property is unknown. However, for alternating-time temporal logic under imperfect information and perfect recall (Belardinelli, Ferrando, and Malvone 2023) and alternating-time temporal logic with knowledge (Lomuscio and Michaliszyn 2016), the abstraction-refinement framework cannot guarantee that a complete abstract model is obtained since finding failure states in these two logics is undecidable. The key difference between the above works and our paper is that they focuses on abstraction on models while we investigate abstraction on theories.

An abstraction framework based on the situation calculus and ConGolog programming language was proposed in (Banihashemi, De Giacomo, and Lespérance 2017). Later, this abstraction framework was extended to cope with online execution with sensing actions (Banihashemi, De Giacomo, and Lespérance 2018), probabilistic domains (Belle 2020; Hofmann and Belle 2023), generalized planning (Cui, Liu, and Luo 2021), non-deterministic actions (Banihashemi, De Giacomo, and Lespérance 2023), and multi-agent games (Lespérance et al. 2024). Cui, Kuang, and Liu (2023) explored the automatic verification of sound abstractions for generalized planning. However, none of these aforementioned works investigates the computation problem of abstractions. Luo et al. (2020) showed that sound and complete abstractions can be characterized theoretically via the notion of forgetting. However, forgetting is a computationally difficult problem. Hence, the above method is not a practical approach.

Conclusions and Future Work

In this paper, we develop a syntactic approach to computing complete and sound abstraction in the situation calculus. To this end, we first present a variant of situation calculus, namely linear integer situation calculus, using LIA as the underlying logic. We then migrate Banihashemi, De Giacomo, and Lespérance (2017)'s abstraction framework to one from linear integer situation calculus to extended situation calculus with the following modifications: (1) The high-level BAT is based on linear integer situation calculus allowing functional fluents. (2) The refinement mapping maps every high-level action to a closed low-level Golog program, maps every high-level predicate fluent to a closed low-level formula, and maps every high-level functional fluent to a closed low-level counting term. In addition, we impose six restrictions (flatness, completeness, simple forgettability, consistency, syntax-irrelevancy and propositional $\exists$-definability) on refinement mapping. We also define a translation m^{-1} from low-level propositional $\exists$-formulas to high-level formulas. Under refinement mappings with the above restrictions, initial KB, action precondition axioms and successor state axioms of high-level BAT can be syntactically constructed from the corresponding low-level counterparts via the translation m^{-1}.

This paper opens several avenues for further work. (1) While this paper provides theoretical results on abstraction, it is interesting to implement the syntactic approach so as to automatically produce high-level action theories. (2) This paper assumes that the restricted refinement mapping is provided. We will explore how to generate suitable refinement mappings from low-level action theories. (3) The four restrictions: completeness, simple-forgettability, consistency and propositional $\exists$-definability are not syntactical. One possible direction is to develop an automatic method to verify if a refinement mapping satisfies the above four restrictions. (4) We plan to loosen the restrictions in this paper so as to devise a more general syntactic approach to suit more complex planning domains.

Acknowledgments

We are grateful to Yongmei Liu, Hong Dong and Shiguang Feng for their helpful discussions on the paper. This paper was supported by National Natural Science Foundation of China (Nos. 62276114, 62077028, 62206055 and 62406108), Guangdong Basic and Applied Basic Research Foundation (Nos. 2023B1515120064, 2024A1515011762), Guangdong-Macao Advanced Intelligent Computing Joint Laboratory (No. 2020B1212030003), Science and Technology Planning Project of Guangzhou (No. 2025A03J3565), Research Foundation of Education Bureau of Hunan Province of China (No. 23B0471), the Fundamental Research Funds for the Central Universities, JNU (No.

21623202).

References

Ball, T.; and Kupferman, O. 2006. An Abstraction-refinement Framework for Multi-Agent Systems. In *Proceedings of the Twenty-First Annual IEEE Symposium on Logic in Computer Science (LICS-2006)*, 379–388.

Banihashemi, B.; De Giacomo, G.; and Lespérance, Y. 2017. Abstraction in Situation Calculus Action Theories. In *Proceedings of Thirty-First AAAI Conference on Artificial Intelligence (AAAI-2017)*, 1048–1055.

Banihashemi, B.; De Giacomo, G.; and Lespérance, Y. 2018. Abstraction of Agents Executing Online and their Abilities in the Situation Calculus. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-2018)*, 1699–1706.

Banihashemi, B.; De Giacomo, G.; and Lespérance, Y. 2023. Abstraction of Nondeterministic Situation Calculus Action Theories. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI-2023)*, 3112–3122.

Belardinelli, F.; Ferrando, A.; Jamroga, W.; Malvone, V.; and Murano, A. 2023. Scalable Verification of Strategy Logic through Three-Valued Abstraction. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI-2023)*, 46–54.

Belardinelli, F.; Ferrando, A.; and Malvone, V. 2023. An abstraction-refinement framework for verifying strategic properties in multi-agent systems with imperfect information. *Artificial Intelligence*, 316: 103847.

Belardinelli, F.; Lomuscio, A.; and Michaliszyn, J. 2016. Agent-based Refinement for Predicate Abstraction of Multi-agent Systems. In *Proceedings of the Twenty-Second European Conference on Artificial Intelligence (ECAI-2016)*, 286–294.

Belle, V. 2020. Abstracting probabilistic models: Relations, constraints and beyond. *Knowledge Based Systems*, 199: 105976:1–15.

Bonet, B.; Francès, G.; and Geffner, H. 2019. Learning Features and Abstract Actions for Computing Generalized Plans. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-2019)*, 2703–2710.

Bonet, B.; Fuentetaja, R.; E-Martín, Y.; and Borrajo, D. 2019. Guarantees for Sound Abstractions for Generalized Planning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-2019)*, 1566–1573.

Boole, G. 1854. *An Investigation of the Laws of Thought*. Walton & Maberly.

Cook, S. A.; and Liu, Y. 2003. A Complete Axiomatization for Blocks World. *Journal of Logic and Computation*, 13(4): 581–594.

Cooper, D. C. 1972. Theorem Proving in Arithmetic without Multiplication. *Machine Intelligence*, 300.

Cui, Z.; Kuang, W.; and Liu, Y. 2023. Automatic Verification for Soundness of Bounded QNP Abstractions for Generalized Planning. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI-2023)*, 3149–3157.

Cui, Z.; Liu, Y.; and Luo, K. 2021. A Uniform Abstraction Framework for Generalized Planning. In *Proceedings of Thirtieth International Joint Conferences on Artificial Intelligence (IJCAI-2021)*, 1837–1844.

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

Grädel, E. 1991. On Transitive Closure Logic. In *Proceedings of the Fifth Workshop on Computer Science Logic (CSL-1991)*, volume 626 of *Lecture Notes in Computer Science*, 149–163. Springer.

Grumberg, O.; Lange, M.; Leucker, M.; and Shoham, S. 2007. When not losing is better than winning: Abstraction and refinement for the full μ-calculus. *Information and Computation*, 205: 1130–1148.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces. *Jourla of the ACM*, 61(3): 16:1–16:63.

Hofmann, T.; and Belle, V. 2023. Abstracting Noisy Robot Programs. In *Proceedings of the Twenty-Second International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2023)*, 534–542.

Kuske, D.; and Schweikardt, N. 2017. First-Order Logic with Counting. In *Proceedings of the Thirty-Second Annual ACM/IEEE Symposium on Logic in Computer Science (LICS-2017)*, 1–12.

Lespérance, Y.; De Giacomo, G.; Rostamigiv, M.; and Khan, S. M. 2024. Abstraction of Situation Calculus Concurrent Game Structures. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI-2024)*, 10624–10634.

Levesque, H. J. 2005. Planning with Loops. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-2005)*, 509–515.

Levesque, H. J.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. B. 1997. GOLOG: A Logic Programming Language for Dynamic Domains. *Journal of Logic Programming*, 59–83.

Li, J.; and Liu, Y. 2020. Automatic Verification of Liveness Properties in the Situation Calculus. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-2020)*, 2886–2892.

Lin, F.; and Reiter, R. 1994. Forget It! In *Proceedings of AAAI Fall Symposium on Relevance*, 154–159.

Lin, X.; Chen, Q.; Fang, L.; Guan, Q.; Luo, W.; and Su, K. 2022. Generalized Linear Integer Numeric Planning. In *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS-2022)*, 241–251.

Lomuscio, A.; and Michaliszyn, J. 2014. An Abstraction Technique for the Verification of Multi-Agent Systems Against ATL Specifications. In *Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning (KR-2014)*, 428–437.

Lomuscio, A.; and Michaliszyn, J. 2016. Verification of Multi-agent Systems via Predicate Abstraction against ATLK Specifications. In *Proceedings of the Fifteenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2016)*, 662–670.

Luo, K.; Liu, Y.; Lespérance, Y.; and Lin, Z. 2020. Agent Abstraction via Forgetting in the Situation Calculus. In *Proceedings of the Twenty-Fourth European Conference on Artificial Intelligence (ECAI-2020)*, 809–816.

Milner, R. 1971. An Algebraic Definition of Simulation Between Programs. In *Proceedings of the Second International Joint Conference on Artificial Intelligence (IJCAI-1971)*, 481–489.

Monniaux, D. 2010. Quantifier Elimination by Lazy Model Enumeration. In *Proceedings of the Twenty-Second International Conference on Computer Aided Verification (CAV-2010)*, volume 6174 of *Lecture Notes in Computer Science*, 585–599. Springer.

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

Sagiv, M.; Reps, T.; and Wilhelm, R. 2002. Parametric Shape Analysis via 3-Valued Logic. *ACM Transactions on Programming Languages and Systems*, 24(3): 217–298.

Segovia-Aguas, J.; Jiménez, S.; and Jonsson, A. 2018. Computing Hierarchical Finite State Controllers with classical planner. *Journal of Artificial Intelligence Research*, 62: 755–797.

Seipp, J.; and Helmert, M. 2018. Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning. *Journal of Artificial Intelligence Research*, 62: 535–577.

Shoham, S.; and Grumberg, O. 2007. A Game-Based Framework for CTL Counterexamples and 3-Valued Abstraction-Refinement. *ACM Transactions on Computation Logic*, 9(1): 1:1–1:52.

Slaney, J.; and Thiébaux, S. 2001. Blocks World revisited. *Artificial Intelligence*, 125(1): 119–153.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2011. A new representation and associated algorithms for generalized planning. *Artificial Intelligence*, 175(2): 615–647.

Srivastava, S.; Zilberstein, S.; Gupta, A.; Abbeel, P.; and Russell, S. 2015. Tractability of Planning with Loops. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-2015)*, 3393–3401.

Srivastava, S.; Zilberstein, S.; Immerman, N.; and Geffner, H. 2011. Qualitative Numeric Planning. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI-2011)*, 1010–1016.