Situation Calculus Temporally Lifted Abstractions for Program Synthesis (Extended Abstract)

Giuseppe De Giacomo[a,c], **Yves Lespérance**[b] **and Matteo Mancanelli**[c,*]

[a]University of Oxford, Oxford, UK
[b]York University, Toronto, ON, Canada
[c]Sapienza University, Rome, Italy

Abstract. We address a program synthesis task where one wants to provide a description of the behavior of common data structures and automatically synthesize a concrete program from a proper abstraction. Our framework is based on nondeterministic situation calculus, extended with LTL trace constraints. We propose a notion of *temporally lifted abstraction* to address the scenario in which we have a single high-level action theory/model with incomplete information and nondeterministic actions and a concrete action theory with several models and complete information. LTL formulas are used to specify the temporally extended goals as well as assumed trace constraints. We show that if the agent has a strategy to achieve a goal under trace constraints at the high level, then there is a refinement of the strategy to achieve the refinement of the goal at the low level.

1 Introduction

Program synthesis is the problem of generating a program that accomplishes a task given its specification. In this paper, we focus on tasks that involve the manipulation of data structures such as lists, trees, and graphs. [2] proposed an approach for solving such problems based on generalized planning, i.e., planning for solving multiple instances at once. Inspired by this work, we present a framework based on the situation calculus [7] that allows one to formally specify a low-level (LL) data structure specification and a high-level (HL) abstraction and associated LTL trace constraints, and prove that a strategy synthesized for the abstract theory can be refined into one that achieves the goal at the LL. Our framework is based on the *nondeterministic situation calculus* [3] where each agent action is accompanied by an environment reaction e, written $A(\vec{x}, e)$. A *nondeterministic basic action theory* (NDBAT) can be seen as a special kind of BAT. Since we are interested in imposing some LTL trace constraints to filter the possible world evolution, we leverage on the axiomatization of infinite paths introduced by [4]. A path p is a sequence of situations, and we use $OnPath(p, s)$, $Starts(p, s)$ and $Suffix(p', p, s)$ with their intuitive meaning. Given an LTL trace constraint ψ, we define $Holds(\psi, p)$ (meaning that ψ holds on path p):

$$
\begin{array}{l}
Holds(\phi, p) \doteq \exists s.Starts(p, s) \wedge \phi[s] \\
Holds(\neg\psi, p) \doteq \neg Holds(\psi, p) \\
Holds(\psi_1 \vee \psi_2, p) \doteq Holds(\psi_1, p) \vee Holds(\psi_2, p) \\
Holds(\bigcirc\psi, p) \doteq \exists s, a, s', p'.Starts(p, s) \wedge \\
\quad s' = do(a, s) \wedge Suffix(p', p, s') \wedge Holds(\psi, p') \\
Holds(\psi_1 \,\mathcal{U}\, \psi_2, p) \doteq \exists s, s', p'.Starts(p, s) \wedge \\
\quad s \preceq s' \wedge Suffix(p', p, s') \wedge Holds(\psi_2, p') \wedge \forall s'', p''. \\
\quad (s \preceq s'' \prec s' \wedge Suffix(p'', p, s'')) \supset Holds(\psi_1, p'')
\end{array}
$$

* Presenting and Corresponding Author.
Email: mancanelli@diag.uniroma1.it.

2 Temporally Lifted Abstractions

To address the program synthesis problem, we focus on the case of a (typically propositional) HL action theory/model with nondeterministic actions, which abstracts over a concrete LL action theory with multiple models. At the LL, in each model we have complete information about the data structure's state, while at the HL, we have actions that may have several outcomes and also LTL formulas that constrain the possible traces. In this section, we want to define a notion of *temporally lifted abstraction* for such theories. We first extend the notion of refinement mapping for NDBATs from [1] to handle the LTL trace constraints. They define a mapping m as a triple $\langle m_a, m_s, m_f \rangle$ where m_a associates each HL primitive action A to a ConGolog agent program δ_A^{ag}, m_s associates each A to a ConGolog system program δ_A^{sys}, and m_f maps each situation-suppressed HL fluent $F(\vec{x})$ to a formula ϕ_F. Now, let ψ be an LTL trace constraint and $Hlc(s)$ a LL symbol that signals if a HL action is completed in s. We revisit the definition of m including a new component m_t, which specifies how HL trace constraints are mapped to the LL:

$$
\begin{array}{l}
m_t(\psi) \doteq m_f(\psi) \\
m_t(\neg\psi) \doteq \neg m_t(\psi) \\
m_t(\psi_1 \vee \psi_2) \doteq m_t(\psi_1) \vee m_t(\psi_2) \\
m_t(\bigcirc\psi) \doteq \bigcirc(\neg Hlc\,\mathcal{U}(Hlc \wedge m_t(\psi))) \\
m_t(\psi_1 \,\mathcal{U}\, \psi_2) \doteq (Hlc \supset m_t(\psi_1))\,\mathcal{U}(Hlc \wedge m_t(\psi_2))
\end{array}
$$

As in [5] and [1], respectively, we require the following two constraints to hold: *(i)* $\mathcal{D}_l \cup C \models Hlc(s)$ iff there exists a HL system action sequence $\vec{\alpha}$ such that $\mathcal{D}_l \cup C \models Do(m(\vec{\alpha}), S_0, s)$, and *(ii)* (proper refinement mapping) for every HL system action sequence $\vec{\alpha}$ and every HL action A, $\mathcal{D}_l \cup \mathcal{C} \models \forall s.(Do(m_s(\vec{\alpha}), S_0, s) \supset \forall \vec{x}, s'.(Do_{ag}(m_a(A(\vec{x})), s, s') \equiv \exists e.Do(m_s(A(\vec{x}, e)), s, s'))$.

To relate the HL and the LL NDBATs through an abstraction, [1] exploits the definition of m-isomorphic situations and a variant of bisimulation. In our framework, we stick to a unidirectional version called m-simulation. In particular, a situation s_h in M_h is m-isomorphic to situation s_l in M_l if and only if $M_h, v[s/s_h] \models F(\vec{x}, s)$ iff $M_l, v[s/s_l] \models m(F(\vec{x}))[s]$ for every HL fluent $F(\vec{x})$ and every assignment v. A relation $R \subseteq \Delta_S^{M_h} \times \Delta_S^{M_l}$ is an m-simulation relation between M_h and M_l if $\langle s_h, s_l \rangle \in R$ implies *(i)* s_h is m-isomorphic to s_l, and *(ii)* for every HL sys-

tem action A, if there exists s'_l such that $M_l, v[s/s_l, s'/s'_l] \models Do(m(A(\vec{x})), s, s')$, then there exists s'_h such that $M_h, v[s/s_h, s'/s'_h] \models Poss(A(\vec{x}), s) \wedge s' = do(A(\vec{x}), s)$ and $\langle s'_h, s'_l \rangle \in R$. M_h is m-similar to M_l wrt the mapping m iff there exists an m-simulation R s.t. $\langle S_0^{M_h}, S_0^{M_l} \rangle \in R$. Having an m-simulation means that if a refinement of a HL action can occur, so can the HL action. Finally, we can present the concept of temporally lifted abstractions:

Definition 1 (Temporally Lifted Abstraction)**.** *Consider an HL NDBAT $\mathcal{D}_h$ equipped with a set of HL state constraint Ψ, a model M_h of $\mathcal{D}_h$, a LL NDBAT $\mathcal{D}_l$ and a refinement mapping m. We say that $(\mathcal{D}_h, M_h, \Psi)$ is a temporally lifted abstraction wrt m if and only if*

- *M_h m-simulates every model M_l of D_l*
- *for every $\psi \in \Psi$, $M_h \models \exists p_h.Starts(p_h, S_{0_h}) \wedge Holds(\psi, p_h)$ and $D_l \models \forall p_l.Starts(p_l, S_{0_l}) \supset Holds(m_t(\psi), p_l)$*

Example. Consider the task of finding the minimum value stored in a singly-linked list. We can construct a NDBAT $\mathcal{D}_l^{sll}$ to capture the behavior of a list with the following fluents: $sll(head, s)$, to identify the head of the list; $node(node\_id, value, next\_id, s)$, to store the nodes; $iterator(cursor, s)$, to represent a cursor; $min\_register(value, s)$, to store the minimum value. The actions for our task are it_set, which creates an iterator, it_next, which moves the cursor to the next node, it_get, which writes the value of the current node into the register, and $mark(node\_id, \text{visited})$.

At the HL, NDBAT $\mathcal{D}_h^{sll}$ uses two fluents: $hasNext(s)$, signaling if there exists a successor node, and $lowerThan(s)$, signaling if the node's value is lower than the register's. The needed actions are $next$, which moves the cursor if possible, $checkValue$, which performs the comparison between the node and the register, and $update$, which updates the register. The environment reaction of $next$ tells us if the end is reached, while that of $checkValue$ returns the result of the comparison. Here is a fragment of the mapping m^{sll}:

$m_s(checkValue(r_h)) =$
  if $\neg\exists c.iterator(c)$ **then** $it\_set(Success_{IC})$ **else** nil **endIf**
  $(\pi c).[iterator(c)?; mark(c, \text{visited}, Success_{MD})]$;
  if $\exists c, v, k, v'.iterator(c) \wedge node(c, v, k) \wedge$
    $min\_register(v') \wedge v < v'$
    then $r_h = \text{LT}$? **else** $r_h = \text{GEQ}$? **endIf**

Additionally, we introduce two LL actions $startHLAction$ and $endHLAction$ which make $Hlc(s)$ false and true respectively at the beginning and at the end of the program of each HL action. We also use a HL fluent for each action, namely $doneNext$, $doneUpdate$ and $doneCheckValue$, to signal the last action executed. Finally, we consider the HL trace constraint for specifying that moving repeatedly to the next node of the list eventually leads to the last one:

$$(\Box\Diamond doneNext) \rightarrow \Diamond\neg hasNext$$

It is possible to prove that m^{sll} is proper wrt $\mathcal{D}_l^{sll}$ and that $(\mathcal{D}_h^{sll}, M_h^{sll}, \Psi)$ is a temporally lifted abstraction of $\mathcal{D}_l^{sll}$ wrt m^{sll}.

3 Results about Strategic Reasoning

Now we want to address the problem of synthesis in the context of temporally lifted abstractions, that is, generating strategies to achieve given goals at the abstract and at the concrete level. For NDBATs, a strong plan is a strategy for the agent that guarantees the achievement of a goal no matter how the environment reacts, defined as a function from situations to agent actions, i.e. $f(s) = A(\vec{x})$. We define $AgtCanForceByIf(Goal, Cstr, f, s)$, meaning that the agent can force a LTL $Goal$ to hold no matter how the environment responds by following strategy f in s if LTL path constraint $Cstr$ holds:

$$AgtCanForceByIf(Goal, Cstr, f, s) \doteq CertainlyExecutable(f, s) \wedge \forall p.Out(p, f, s) \wedge Holds(Cstr, p) \supset Holds(Goal, p)$$

where

$$CertainlyExecutable(f, s) \doteq \exists P.[\forall s.P(s) \supset [Poss_{ag}(f(s), s)] \wedge [\forall s'.Do_{ag}(f(s), s, s') \supset P(s')]] \wedge P(s)$$

$$Out(p, f, s) \doteq \forall a.\forall s.OnPath(p, s) \wedge OnPath(p, do(a, s)) \supset Do_{ag}(f(s), s, do(a, s))$$

$CertainlyExecutable(f, s)$ captures the requirement that the strategy never prescribes an action that is not executable; $Out(p, f, s)$ means that path p is a possible outcome of the agent executing strategy f in s. We also define $AgtCanForceIf(Goal, Cstr, s) \doteq \exists f.AgtCanForceByIf(Goal, Cstr, f, s)$.

As in [1], we impose an additional constraint: for every HL action A, there exists a LL strategy f_A such that for every HL $\vec{\alpha}$, $\mathcal{D}_l \models \forall s.Do(m(\vec{\alpha}, S_0, s) \supset (\forall \vec{x}.\exists s'.Do_{ag}(m_a(A(\vec{x})), s, s') \supset AgtCanForceBy(m_a(A(\vec{x})), f_A, s))$. It requires that for any HL action executable at the LL, the agent has a strategy to execute it regardless of the environment. Now, we can prove that, given a temporally lifted abstraction, if the agent has a strategy to achieve a LTL goal under some constraints at the HL, then there exists a refinement of the strategy to achieve the refinement of the goal at the LL.

Theorem 1. *Let $(\mathcal{D}_h, M_h, Cstr)$ be a temporally lifted abstraction of $\mathcal{D}_l$ wrt refinement mapping m s.t. previous constraints hold, and $Goal$ be an LTL goal. Then we have that:*

if $M_h \models AgtCanForceIf(Goal, Cstr, S_0)$,
then there exist a LL strategy f_l such that
$\mathcal{D}_l \models AgtCanForceByIf(m(Goal), \mathit{True}, f_l, S_0)$

Example Cont. To find the minimum value in a singly-linked list the HL LTL goal constraints that must be satisfied are:

$$\Diamond\Box\neg hasNext$$
$$\Box(doneNext \rightarrow \bigcirc doneCheckValue)$$
$$\Box(lowerThan \leftrightarrow \bigcirc doneUpdate)$$

In [2], authors show how to use LTL synthesis engines like Strix [6] to get a controller policy for this task. Here is a strategy that guarantees the satisfaction of the goals: $f_h(s) = checkValue$ if $doneNext(s)$; $f_h(s) = update$ if $lowerThan(s)$; $f_h(s) = next$ if $\neg lowerThan(s) \wedge hasNext(s)$; $f_h(s) = stop$ otherwise.

It is easy to derive a refined LL strategy and apply Theorem 1 to show that $M_h \models AgtCanForceByIf(Goal, Cstr, f_h, S_0)$ and $\mathcal{D}_l \models AgtCanForceBy(m(Goal), \mathit{True}, f_l, S_0)$.

References

[1] B. Banihashemi, G. De Giacomo, and Y. Lespérance. Abstraction of nondeterministic situation calculus action theories. 3:3112–3122, 2023.

[2] B. Bonet, G. De Giacomo, H. Geffner, F. Patrizi, and S. Rubin. High-level programming via generalized planning and LTL synthesis. In *KR 2020*, volume 17, pages 152–161.

[3] G. De Giacomo and Y. Lespérance. The nondeterministic situation calculus. In *KR 2021*, volume 18.

[4] S. M. Khan and Y. Lespérance. Infinite paths in the situation calculus: Axiomatization and properties. In *KR 2016*.

[5] Y. Lespérance, G. D. Giacomo, M. Rostamigiv, and S. M. Khan. Abstraction of situation calculus concurrent game structures. In *AAAI 2024*.

[6] P. J. Meyer, S. Sickert, and M. Luttenberger. Strix: Explicit reactive synthesis strikes back! Lecture Notes in Computer Science. Springer, 2018.

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