Situation Calculus Temporally Lifted Abstractions for Generalized Planning

Giuseppe De Giacomo[1,2], **Yves Lespérance**[3], **Matteo Mancanelli**[2]
[1]University of Oxford, Oxford, UK
[2]University of Rome La Sapienza, Rome, Italy
[3]York University, Toronto, ON, Canada
giuseppe.degiacomo@cs.ox.ac.uk, lesperan@eecs.yorku.ca, mancanelli@diag.uniroma1.it

Abstract

We present a new formal framework for generalized planning (GP) based on the situation calculus extended with LTL constraints. The GP problem is specified by a first-order basic action theory whose models are the problem instances. This low-level theory is then abstracted into a high-level propositional nondeterministic basic action theory with a single model. A refinement mapping relates the two theories. LTL formulas are used to specify the temporally extended goals as well as assumed trace constraints. If all LTL trace constraints hold at the low level and the high-level model can simulate all the low-level models with respect to the mapping, we say that we have a *temporally lifted abstraction*. We prove that if we have such an abstraction and the agent has a strategy to achieve a LTL goal under some trace constraints at the abstract level, then there exists a refinement of the strategy to achieve the refinement of the goal at the concrete level. We use LTL synthesis to generate the strategy at the abstract level. We illustrate our approach by synthesizing a program that solves a data structure manipulation problem.

Introduction

In generalized planning (GP), one tries to generate a typically iterative policy that solves an infinite set of similar planning problem instances (Srivastava, Immerman, and Zilberstein 2008; Hu and De Giacomo 2011; Belle and Levesque 2016). For example, we may want to synthesize a program for finding the minimum value in a list, for lists of any lengths. Many approaches to GP involve constructing an abstraction and finding a solution for this abstraction which handles all the actual problem instances.

We propose a new formal framework for GP based on the situation calculus (McCarthy and Hayes 1969; Reiter 2001) that allows one to provide an abstract description of the domain and associated LTL trace constraints (Bonet et al. 2017; Aminof et al. 2019), and prove that a controller synthesized for the abstract theory can be refined into one that achieves the goal at the concrete level.

Our framework is based on the *nondeterministic situation calculus* (De Giacomo and Lespérance 2021) (DL21), where each agent action is accompanied by an environment reaction outside the agent's control that determines the action's outcome, e.g., a flipped coin may fall head or tail. (Banihashemi, De Giacomo, and Lespérance 2023) (BDL23) have proposed an account of abstraction for nondeterministic basic action theories (NDBATs) in this language. They relate a high-level NDBAT to a low-level NDBAT through a *refinement mapping* that specifies how a high-level action is implemented by a ConGolog program at the low level and how a high-level fluent can be defined by low-level state formula. They then define notions of sound and/or complete abstraction for such NDBATs in terms of a notion of bisimulation wrt such a mapping between their models. (Cui, Liu, and Luo 2021; Cui, Kuang, and Liu 2023) have adapted this kind of approach to solve GP, focusing on QNP abstractions.

Here, we assume that the modeler specifies a propositional high-level (HL) action theory/model with a limited set of HL fluents and nondeterministic actions, which abstracts over a concrete low-level (LL) action theory with multiple models, with a given refinement mapping m. At the LL, in each model we have complete information about the state of the world, while at the HL, we have actions that may have several outcomes, e.g., after advancing to the next item in a list, we may or may not reach the list's end. But we also have some HL LTL trace constraints, e.g., ensuring that if we keep advancing we will eventually reach the list's end. We define a notion of *temporally lifted abstraction* for such theories, where every LL trace that is a refinement of a sequence of HL actions is an m-simulation of a trace involving this action sequence in the HL model, and where the LTL trace constraints are satisfied by the LL theory. The NDBATs represent our GP problem, where each LL model specifies the planning problem instances, and the HL model abstracts away the LL details, retaining only the shared features. We then provide a method for solving all the planning problem instances simultaneously. In particular, we show that given such an abstraction, if we can use LTL synthesis on the HL model to obtain a HL strategy to achieve a LTL goal under the given trace constraints, then we can automatically refine it to get a LL strategy that achieves the mapped LTL goal in all concrete instances of the problem.

We illustrate how our approach works by using it to synthesize a program to find the minimum value of a list. This application is inspired by (Bonet et al. 2020) (B20), which proposed an approach for solving program synthesis tasks (Green 1969; Waldinger and Lee 1969; Church 1963; Abadi, Lamport, and Wolper 1989; Pnueli and Rosner 1989) that involve the manipulation of data structures such as lists, trees, and graphs by viewing them as instances of GP. They provide several examples of how their method can be applied, but they do not provide complete formal specifications of the data structures used and formal proofs that the assumed temporal constraints and goal specifications hold for them.

Preliminaries

Nondeterministic situation calculus. The *situation calculus* is a well known predicate logic language designed for representing and reasoning about dynamically changing worlds (McCarthy and Hayes 1969; Reiter 2001). All changes to the world are the result of *actions*, which are terms in the language. A possible world history is represented by a term called a *situation*, which is a sequence of actions. The constant S_0 is used to denote the initial situation, and the function $do(a, s)$ is used to denote the successor situation resulting from performing action a in situation s. Predicates and functions whose value varies from situation to situation are called *fluents*, and are denoted by symbols taking a situation term as their last argument. In this language, a dynamic domain can be represented by a *basic action theory (BAT)*, where successor state axioms (SSAs) represent the causal laws of the domain (Reiter 2001). A predicate *Poss*(a, s) is used to state that a is executable in s; the precondition axioms characterize this predicate.

(DL21) propose a simple extension of the standard situation calculus to handle nondeterministic actions. For any primitive action by the agent in a nondeterministic domain, there can be a number of different outcomes, depending on how the environment reacts to the agent's action. This is modeled by having every action type/function $A(\vec{x}, e)$ take an additional environment reaction parameter e, ranging over a new sort *Reaction*. We call the reaction-suppressed version of the action $A(\vec{x})$ an *agent action* and the full version $A(\vec{x}, e)$ a *system action*.

A *nondeterministic basic action theory* can be seen as a special kind of BAT, and it is the union of the following disjoint sets: foundational, domain independent, axioms of the situation calculus (Σ), axioms describing the initial situation ($\mathcal{D}_{S_0}$), unique name axioms for actions ($\mathcal{D}_{una}$), successor state axioms describing how fluents change after *system* actions are performed ($\mathcal{D}_{ssa}$), and *system* action precondition axioms, one for each action type, stating when the complete system action can occur ($\mathcal{D}_{poss}$). One also specifies agent action preconditions using $Poss_{ag}$. The theory must entail the *reaction independence* requirement (formally, $\forall e.Poss(A(\vec{x}, e), s) \supset Poss_{ag}(A(\vec{x}), s)$) and the *reaction existence* requirement (formally, $Poss_{ag}(A(\vec{x}), s) \supset \exists e.Poss(A(\vec{x}, e), s)$).

High-level programs and ConGolog. To specify how a HL action is implemented at the LL, we focus on (a variant of) ConGolog (De Giacomo, Lespérance, and Levesque 2000), an *HL programming language* characterized by

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

where α is an action term, $\delta_1;\delta_2$ is the sequential execution of δ_1 and δ_2, $\delta_1|\delta_2$ is the nondeterministic choice of δ_1 and δ_2, $\pi x.\delta$ executes program δ for *some* nondeterministic choice of the object variable x, δ^* performs δ zero or more times, and $\delta_1\|\delta_2$ is the interleaved execution of δ_1 and δ_2. Note that φ is a situation-suppressed formula, and we denote by $\varphi[s]$ the formula obtained by restoring the situation argument s into all fluents in φ. Conditional and while-loop constructs are definable as follows: **if** ϕ **then** δ_1 **else** δ_2 **endIf** $= \phi?;\delta_1|\neg\phi?;\delta_2$ and **while** ϕ **do** δ **endWhile** $= (\phi?;\delta)^*;\neg\phi?$. We also use the abbreviation $nil = True?$.

Formally, the semantics of ConGolog is specified in terms of single-step transitions, using the following two predicates: *(i)* $Trans(\delta, s, \delta', s')$, which holds if one step of program δ in situation s may lead to situation s' with δ' remaining to be executed; and *(ii)* $Final(\delta, s)$, which holds if program δ may legally terminate in situation s. The definitions of $Trans$ and $Final$ we use are as in (De Giacomo, Lespérance, and Pearce 2010); differently from (De Giacomo, Lespérance, and Levesque 2000), the test construct $\varphi?$ does not yield any transition, but is final when satisfied. Predicate $Do(\delta, s, s')$ means that program δ, when executed starting in situation s, has as a legal terminating situation s', and is defined as $Do(\delta, s, s') \doteq \exists\delta'.Trans^*(\delta, s, \delta', s') \wedge Final(\delta', s')$, where *Trans** denotes the reflexive transitive closure of *Trans*. We use $\mathcal{C}$ to denote the axioms defining the ConGolog programming language.

For simplicity, we use a restricted class of ConGolog programs that are *situation-determined* (SD) (De Giacomo, Lespérance, and Muise 2012), i.e., for every action sequence, the remaining program is uniquely determined by the resulting situation: $SitDet(\delta, s) \doteq \forall s', \forall\delta', \delta''.$ *Trans*$^*(\delta, s, \delta', s') \wedge$ *Trans*$^*(\delta, s, \delta'', s') \supset \delta' = \delta''$

Generalized Planning. A generalized planning problem is a (finite or infinite) set of basic planning problems that share the same set of actions and features (or observations) (Hu and De Giacomo 2011). Intuitively, solving a GP problem means finding a strategy that serves as a solution for each of the basic planning problems. A general way to represent a GP problem is to project the basic planning problems onto their common observation space, resulting in an observation abstraction (Bonet and Geffner 2015; Bonet et al. 2017). Similarly to (Hu and Levesque 2010; De Giacomo et al. 2016; Cui, Liu, and Luo 2021), in our framework a GP is defined as a pair of an action theory and a (LTL) goal. Specifically, each model of the action theory represents a basic planning problem, and we abstract them by using a propositional HL action theory, where we have a single model and nondeterministic actions. The propositional fluents in each situation represent the common observations among all the LL models, and the outcome of an action depends on the environment's reaction, which models the possible different ways in which LL traces proceed. We extend the HL theory with trace constraints to impose fairness assumptions on the possible sequence of nondeterministic actions. A strategy at the HL must ensure that the agent achieves the HL goal under the trace constraints, regardless of how the environment behaves. This strategy is a solution to the GP problem if it can be refined at the LL and ensures that the agent achieves the LL goal for every model.

Linear Temporal Logic. Linear Temporal Logic (LTL) is one of the most popular formalisms for expressing temporal properties of reactive systems (Pnueli 1977). Given a set of atomic propositions P the formulas of LTL are as follows:

$$\phi ::= a \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \bigcirc\phi \mid \phi_1 \mathcal{U} \phi_2$$

where $a \in P$. We use common abbreviations such as *eventually* as $\Diamond\phi \doteq \mathit{True}\,\mathcal{U}\,\phi$ and *always* as $\Box\phi \doteq \neg\Diamond\neg\phi$.

Formulas of LTL are interpreted over infinite sequences (called traces) of truth evaluations of variables in P , i.e. $\pi = \pi_0, \pi_1, \cdots \in (2^P)^\infty$. Given a trace π, we define when an LTL formula ϕ holds at position i on π, written $\pi, i \models \phi$, inductively on the structure of ϕ, as follows:

- $\pi, i \models a$ iff $a \in \pi_i$ (for $a \in P$);
- $\pi, i \models \neg\phi$ iff $\pi, i \not\models \phi$;
- $\pi, i \models \phi_1 \vee \phi_2$ iff $\pi, i \models \phi_1$ or $\pi, i \models \phi_2$;
- $\pi, i \models \bigcirc\phi$ iff $\pi, i+1 \models \phi$;
- $\pi, i \models \phi_1 \mathcal{U} \phi_2$ iff there exists $j \geq i$ such that $\pi, j \models \phi_2$, and for all k, $i \leq k < j$ we have that $\pi, k \models \phi_1$.

We say that π satisfies ϕ, written $\pi \models \phi$, if $\pi, 0 \models \phi$.

LTL constraints in NDBATs. We are interested in imposing some LTL temporal properties that can work as trace constraints to filter the considered world histories in the context of NDBATs. A convenient way for doing this is to leverage the axiomatization of infinite paths introduced by (Khan and Lespérance 2016), which provided a natural way to talk about "infinite future histories". An infinite sequence of situations is called a path, and we have a special sort *paths* and a predicate $OnPath(p, s)$ meaning that situation s is on path p. Additionally, we will use $Starts(p, s)$, to specify that the path p starts with situation s, and $Suffix(p', p, s)$, which means that the path p' starts with s and contains the same situations as p starting from s.

Based on this, we define a special predicate $Holds(\psi, p)$ to specify that a given LTL property ψ holds on path p. Here, we assume that the LTL atomic propositions ϕ are ground situation-suppressed formulas defined over an NDBAT.

Definition 1. *If ψ is an LTL trace constraint and p is an infinite path, we define $Holds(\psi, p)$ (meaning that the constraint ψ holds on path p) inductively as follows*[1]*:*

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

Note that the original axiomatization of infinite paths does not consider nondeterministic actions but it can be easily extended to NDBATs by focusing on system actions.

[1] $s \prec s'$ denotes that the actions performed to reach s' from s were all executable; $s \preceq s'$ stands for $s \prec s' \vee s = s'$

Temporally Lifted Abstractions

We want to define a GP problem at the concrete level using a BAT $\mathcal{D}_l$, where models of this BAT serve as the basic planning problem instances of the GP. We will then define an abstract theory by providing a propositional NDBAT $\mathcal{D}_h$, for which we have a single model and where nondeterministic effects reflect the variability among different concrete instances. We will also impose some LTL trace constraints on the abstract model to characterize the traces that can actually occur in the HL action sequences. We refer to this as a temporally lifted abstraction. With this structure in place, we can approach solving the GP problem by performing LTL synthesis on the abstraction.

For this to work, we need to ensure that executions of refinements of HL actions in models of the LL theory correspond to executions in the HL theory/model. We will specify the relationship between the HL NDBAT and the LL BAT by a refinement mapping m (viewing the LL theory as a NDBAT). For this, we extend the notion of refinement mapping for NDBAT abstractions from (BDL23) to handle LTL trace constraints. We will then ensure that executions of HL actions in the models correspond through a form of simulation relative to the refinement mapping m.

NDBAT Refinement Mapping with Trace Constraints. In (BDL23), an NDBAT refinement mapping m is a triple $\langle m_a, m_s, m_f\rangle$ where

- m_a associates each HL primitive action type A to a ConGolog agent program δ_A^{ag} defined over the LL theory that implements the agent action (i.e., $m_a(A(\vec{x})) = \delta_A^{ag}(\vec{x})$)
- m_s associates each A to a ConGolog system program δ_A^{sys} defined over the LL theory that implements the system action (i.e., $m_s(A(\vec{x}, e)) = \delta_A^{sys}(\vec{x}, e)$)
- m_f maps each situation-suppressed HL fluent $F(\vec{x})$ to a situation-suppressed formula $\phi_F(\vec{x})$ defined over the LL theory that characterizes the conditions under which $F(\vec{x})$ holds in a situation (i.e., $m_f(F(\vec{x})) = \phi_F(\vec{x})$)

We can extend such a mapping to a sequence of agent actions as follows: $m_a(\alpha_1, \ldots, \alpha_n) \doteq m_a(\alpha_1); \ldots; m_a(\alpha_n)$ for $n \geq 1$ and $m_a(\epsilon) \doteq nil$, and similarly for system actions sequences. We also extend the notation so that $m_f(\phi)$ stands for the result of substituting every fluent $F(\vec{x})$ in situation-suppressed formula ϕ by $m_f(F(\vec{x}))$.

A refinement mapping must comply with the following:

Constraint 2 (Proper Refinement Mapping)**.** *For every HL system action sequence $\vec{\alpha}$ and every HL action A, we have:*

$$\begin{aligned}
&\mathcal{D}_l \cup \mathcal{C} \models \forall s.(Do(m_s(\vec{\alpha}), S_0, s) \supset \forall \vec{x}, s'.\\
&(Do_{ag}(m_a(A(\vec{x})), s, s') \equiv \exists e.Do(m_s(A(\vec{x}, e)), s, s'))
\end{aligned}$$

It guarantees that *(1)* for every s' reachable from s by a refinement of $A(\vec{x})$, there is a reaction e that generates it, and *(2)* for every s' reachable from s by a refinement of $A(\vec{x}, e)$, there is a refinement of $A(\vec{x})$ that generates it. This ensures consistency between agent and system actions, satisfying the reaction existence and reaction independence requirements.

Here, we want to extend the NDBAT refinement mapping to also handle HL LTL trace constraints, which we want to be able to map to equivalent LL trace constraints. For this, we introduce an additional constraint from (Lespérance et al. 2024) so that the LL theory tracks when refinements of HL actions end using a state formula $Hlc(s)$, meaning that a HL action has just completed in situation s:

Constraint 3. *$\mathcal{D}_l \cup C \models Hlc(s)$ if and only if there exists a HL system action sequence $\vec{\alpha}$ such that $\mathcal{D}_l \cup C \models Do(m(\vec{\alpha}), S_0, s)$.*

There are various ways to define $Hlc(s)$ to satisfy this constraint; see the example for some discussion.

The revisited definition of NDBAT mapping maintains the original elements and adds a new component m_t which specifies how HL trace constraints are mapped to the LL:

Definition 4 (Refinement Mapping for Trace Constraints)**.** *Let ψ be an LTL trace constraint and Hlc a distinguished symbol which signals that a HL action is completed. A NDBAT refinement mapping m is a tuple $\langle m_a, m_s, m_f, m_t\rangle$, where m_a, m_s, and m_f are defined as usual and m_t is a mapping for trace constraints defined as follows:*

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

Here, we say that if $\bigcirc\psi$ holds at the HL, then $m_t(\psi)$ must hold at the LL in the first situation when a HL action has been completed, i.e., Hlc must remain false until it becomes true and at that point $m_t(\psi)$ must hold.

Temporally lifted abstractions. Finally, we can present the concept of temporally lifted abstractions. To relate the HL and LL models/theories, we first define:

Definition 5 (m-isomorphic situations)**.** *Let M_h and M_l be models of $\mathcal{D}_h$ and $\mathcal{D}_l$, respectively. We say that situation s_h in M_h is m-isomorphic to situation s_l in M_l, written $s_h \simeq_m^{M_h,M_l} s_l$, iff*

$$M_h, v[s/s_h] \models F(\vec{x}, s) \text{ iff } M_l, v[s/s_l] \models m_f(F(\vec{x}))[s]$$

for every high-level primitive fluent $F(\vec{x})$ in F_h and every variable assignment v[2].

If $s_h \simeq_m^{M_h,M_l} s_l$, s_h and s_l evaluate all HL fluents the same.

(BDL23) and the earlier (Banihashemi, De Giacomo, and Lespérance 2017) define a variant of bisimulation (Milner 1971, 1989) to establish a relation based on the refinement mapping. Here, we stick to a unidirectional version:

Definition 6 (m-simulation)**.** *Let $\Delta_S^{M_h}$ and $\Delta_S^{M_l}$ be the situation domains of M_h and M_l respectively. 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*

1. *s_h is m-isomorphic to s_l*
2. *for every HL system action A, if there exists s'_l such that $M_l, v[s/s_l, s'/s'_l] \models Do(m_a(A(\vec{x}, e)), s, s')$, then there exists s'_h such that $M_h, v[s/s_h, s'/s'_h] \models Poss(A(\vec{x}, e), s) \wedge s' = do(A(\vec{x}, e), s)$ and $\langle s'_h, s'_l\rangle \in R$*

[2]$v[x/e]$ denotes the variable assignment that is like v except that the variable x is assigned the entity e

We say that M_h is m-similar to M_l wrt the mapping m (written $M_h \sim_m^{\leftarrow} M_l$) iff there exists an m-simulation relation R between M_h and M_l such that $\langle S_0^{M_h}, S_0^{M_l}\rangle \in R$. We also say that a situation s_h in M_h is m-similar to situation s_l in M_l (written $s_h \sim_m^{M_h,M_l,\leftarrow} s_l$) iff there exists an m-simulation relation R between M_h and M_l and $\langle s_h, s_l\rangle \in R$.

Having an m-simulation means that if a refinement of a HL action can occur, so can the HL action. A similar idea is present in (Cui, Liu, and Luo 2021), where the definition of m-bisimulation is decomposed into m-simulation and m-back-simulation to represent the two directions. It is easy to prove the following lemma about m-similar situations.

Lemma 7. *If $s_h \simeq_m^{M_h,M_l,\leftarrow} s_l$, then for any high-level situation-suppressed formula ϕ, we have that:*

$$M_h, v[s/s_h] \models \phi[s] \textit{ iff } M_l, v[s/s_l] \models m_f(\phi)[s]$$

Proof. By induction on the structure of ϕ. □

At last, exploiting m-simulation together with the use of LTL trace constraints, we can talk about the notion of temporally lifted abstractions. Intuitively, we have a temporally lifted abstraction if there is an m-simulation between an HL model/theory and all the models of a LL theory, and every trace constraint is satisfied on some path at the HL and on all paths at the LL.

Definition 8 (Temporally Lifted Abstraction)**.** *Consider an HL NDBAT $\mathcal{D}_h$ equipped with a set of HL trace 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 and $\mathcal{D}_l$ if and only if*

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

We may want to verify that we have a temporally lifted abstraction. To this end, we first show a lemma that identifies sufficient conditions for having an m-simulation. (Complete proofs of all results are available in the extended version.)

Lemma 9. *Suppose that $M_h \models \mathcal{D}^h$ for some high-level theory $\mathcal{D}^h$ and $M_l \models \mathcal{D}^l \cup \mathcal{C}$ for some low-level theory $\mathcal{D}^l$ and m is a mapping between the two theories. If*

(a) $S_0^{M_h} \simeq_m^{M_h,M_l} S_0^{M_l}$

(b) *for all high-level action sequences $\vec{\alpha}$,*

$$\begin{aligned}&\mathcal{D}^l \cup \mathcal{C} \models \forall s.Do(m_s(\vec{\alpha}), S_0, s) \supset\\ &\quad\textstyle\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}.(\exists s' Do(m_s(A_i(\vec{x})), s, s')) \supset\\ &\qquad m_f(\phi_{A_i}^{Poss}(\vec{x}))[s]\end{aligned}$$

where $\mathcal{A}^h$ is the set of actions and $\phi_{A_i}^{Poss}(\vec{x})$ is the right-hand side (RHS) of the precondition axiom for $A_i(\vec{x})$,

(c) *for all high-level action sequences $\vec{\alpha}$,*

$$\begin{aligned}&\mathcal{D}^l \cup \mathcal{C} \models \forall s.Do(m_s(\vec{\alpha}), S_0, s) \supset\\ &\textstyle\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}, s'.(Do(m_s(A_i(\vec{x})), s, s') \supset\\ &\textstyle\bigwedge_{F_i \in \mathcal{F}^h} \forall \vec{y}(m_f(\phi_{F_i,A_i}^{ssa}(\vec{y}, \vec{x}))[s] \equiv m_f(F_i(\vec{y}))[s']))\end{aligned}$$

where $\mathcal{F}^h$ is the set of fluents and $\phi_{F_i,A_i}^{ssa}(\vec{y}, \vec{x})$ is the RHS of the SSA for F_i instantiated with action $A_i(\vec{x})$ where action terms have been eliminated using $\mathcal{D}_{ca}^h$

then $M_h \sim_m^{\leftarrow} M_l$.

Proof (sketch). We assume the antecedent and we need to prove that $M_h \sim_m^{\leftarrow} M_l$. Let R be the relation over $\Delta_S^{M_h} \times \Delta_S^{M_l}$ such that

$\langle s_h, s_l \rangle \in R$
if and only if
there exists a ground high-level action sequence $\vec{\alpha}$
such that $M_l, v[s/s_l] \models Do(m(\vec{\alpha}), S_0, s)$
and $s_h = do(\vec{\alpha}, S_0)^{M_h}$.

We want to show that R is an m-simulation relation between M_h and M_l, i.e. if $\langle s_h, s_l \rangle \in R$, then it satisfies the two conditions for m-simulation in Definition 6. Both conditions can be proven by induction on n, the number of actions in s_h, using Lemma 7. □

Then, one can verify that one has a temporally lifted abstraction using the following theorem:

Theorem 10. *Suppose that we have a HL NDBAT $\mathcal{D}_h$, a model M_h of $\mathcal{D}_h$, a set of HL LTL trace constraints Ψ, a LL NDBAT $\mathcal{D}_l$, and a refinement mapping m. If*

(a) $\mathcal{D}^h_{S_0}$ *is a complete theory, i.e. the initial state is completely specified,*

(b) $\mathcal{D}^l_{S_0} \cup \mathcal{D}^l_{ca} \cup \mathcal{D}^l_{coa} \models m(\phi)$, *for all* $\phi \in \mathcal{D}^h_{S_0}$,

(c) *for all high-level action sequences* $\vec{\alpha}$,

$$\begin{aligned}&\mathcal{D}^l \cup \mathcal{C} \models \forall s. Do(m_s(\vec{\alpha}), S_0, s) \supset \\ &\quad \textstyle\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}. (\exists s' Do(m_s(A_i(\vec{x})), s, s')) \supset \\ &\quad\quad m_f(\phi^{Poss}_{A_i}(\vec{x}))[s]\end{aligned}$$

(d) *for all high-level action sequences* $\vec{\alpha}$,

$$\begin{aligned}&\mathcal{D}^l \cup \mathcal{C} \models \forall s. Do(m_s(\vec{\alpha}), S_0, s) \supset \\ &\quad \textstyle\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}, s'. (Do(m_s(A_i(\vec{x})), s, s') \supset \\ &\quad \textstyle\bigwedge_{F_i \in \mathcal{F}^h} \forall \vec{y} (m_f(\phi^{ssa}_{F_i, A_i}(\vec{y}, \vec{x}))[s] \equiv m_f(F_i(\vec{y}))[s']))\end{aligned}$$

(e) *for every high-level LTL trace constraint* $\psi \in \Psi$,
$M_h \models \exists p_h. Starts(p_h, S_{0_h}) \land Holds(\psi, p_h)$ *and*
$D_l \models \forall p_l. Starts(p_l, S_{0_l}) \supset Holds(m_t(\psi), p_l)$,

then $(\mathcal{D}_h, M_h, \Psi)$ *is a temporally lifted abstraction of* $\mathcal{D}_l$ *wrt* m.

Proof. (a) and (b) imply that for every model M_l of $\mathcal{D}_l$, $S_0^{M_h} \simeq_m^{M_h, M_l} S_0^{M_l}$. Together with (c) and (d), this implies by Lemma 9 that for every model M_l of $\mathcal{D}_l$, $M_h \sim_m^{\leftarrow} M_l$. The result then follows from (e) and the definition of temporally lifted abstraction. □

Example (Minimum in a List). To illustrate our framework, taking inspiration from the work done by (B20), we use common programming problems and data structures. As a first example, we consider the task of finding the minimum value in a singly-linked list. NDBAT $\mathcal{D}^{sll}_l$ is the LL action theory that describes the operations that can be performed and how they affect the predicates representing lists. In it, we mainly use two actions and two fluents.[3] The actions are $next_{LL}$, which moves a cursor that scans the nodes of the list, and $update_{LL}$, which writes the value of the current pointed node into a dedicated register. Additionally, we have a *no_op* action, with no preconditions and no effects. Note that at the LL, we have complete information and deterministic actions, thus we have only one possible environment reaction *Success* for each action. The fluents will be $pos(s)$, whose value represents the position of the current node within the list, and $cmp(s)$, which represents whether the current node contains a lower value than the register. Specifying action precondition axioms and successor state axioms for $\mathcal{D}^{sll}_l$ is straightforward.

[3]The LL theory provided here is not detailed for the sake of accessibility, but it is easy to fully formalize data structure behavior.

At the HL, we have a propositional theory with non-deterministic actions that abstracts over the concrete level and loses some information. Specifically, we don't know the length of the list or the values stored in each node. NDBAT $\mathcal{D}^{sll}_h$ represents the HL domain with two fluents: $hasNext(situation)$, indicating whether the cursor points to the last node of the list, and $lowerThan(situation)$, indicating whether the value of the node pointed at by the cursor is lower than the value stored in the register. Note that, if we want to perform LTL synthesis to solve the task, we need both fluents and actions to be propositional (apart from the situation argument). The actions used for our task will be $next_{HL}$, which moves the cursor and performs the comparison between the node and register values, and $update_{HL}$, which updates the register's value to the node's. In every HL theory, we also introduce the action *stop*, used in infinite paths when the goal is already reached.

The action precondition axioms in $\mathcal{D}^{sll}_h$ are:

$$\begin{aligned}&Poss_{ag}(next_{HL}, s) \doteq hasNext(s)\\ &Poss_{ag}(update_{HL}, s) \doteq True\\ &Poss_{ag}(stop, s) \doteq True\\ &Poss(next_{HL}(r), s) \equiv\\ &\quad Poss_{ag}(next_{HL}, s) \land (r = LT\_NE \lor\\ &\quad\quad r = GEQ\_NE \lor r = LT\_E \lor r = GEQ\_E)\\ &Poss(update_{HL}(r), s) \equiv\\ &\quad Poss_{ag}(update_{HL}, s) \land r = Success_{HLU}\\ &Poss(stop(r), s) \equiv Poss_{ag}(stop, s) \land r = Success_{HS}\end{aligned}$$

Since we have no knowledge of the node components of the list at the HL, we obtain the necessary information via the environment reactions. In particular, the action $next_{HL}$ gathers environment reactions on whether the successor node's value is lower than the register's (LT stands for "lower than"; GEQ stands for "greater than or equal to") and on reaching the end of the list (NE stands for "not end"; E stands for "end"). Thus we have four possible reactions to cover all the possible cases. The SSAs are straightforward:

$$\begin{aligned}&hasNext(do(a, s)) \equiv\\ &\quad a = next_{HL}(\text{LT\_NE}) \lor a = next_{HL}(\text{GEQ\_NE}) \lor\\ &\quad hasNext(s) \land a \neq next_{HL}(\text{LT\_E}) \land\\ &\quad\quad a \neq next_{HL}(\text{GEQ\_E})\\ &lowerThan(do(a, s)) \equiv\\ &\quad a = next_{HL}(\text{LT\_NE}) \lor a = next_{HL}(\text{LT\_E}) \lor\\ &\quad lowerThan(s) \land a \neq next_{HL}(\text{GEQ\_NE}) \land\\ &\quad\quad a \neq next_{HL}(\text{GEQ\_E}) \land\\ &\quad\quad a \neq update_{HL}(Success_{HLU})\end{aligned}$$

We specify the relationship between the HL and LL NDBATs through the following refinement mapping m^{sll}:

$m_a(next_{HL}) = next_{LL}$
$m_a(update_{HL}) = update_{LL}$
$m_a(stop) = no\_op$

```
$m_s(next_{HL}(r_h)) =$
  $next_{LL}(Success_{LLN});$
  **if** $pos < length$
    **if** $cmp =$ LT
      **then** $r_h =$ LT_NE? **else** $r_h =$ GEQ_NE? **endIf**
  **else**
    **if** $cmp =$ LT
      **then** $r_h =$ LT_E? **else** $r_h =$ GEQ_E? **endIf**
  **endIf**
$m_s(update_{HL}(r_h)) =$
  $update_{LL}(Success_{LLU}); r_h = Success_{HLU}?$
$m_s(stop(r_h)) = no\_op(Success_{NO}); r_h = Success_S?$
```

$m_f(hasNext) = pos < length$
$m_f(lowerThan) = (cmp = \text{LT})$

To properly deal with trace constraints with a refinement mapping as described in Definition 4, we need to introduce some additional components to both HL and LL theories. First we specify how to deal with the predicate $Hlc(s)$ that denotes at the LL when an HL action has just completed. A simple way to do that is to introduce two LL actions $startHLAction$ and $endHLAction$ which make $Hlc(s)$ false and true respectively, and place them at the beginning and at the end of the refinement program of each HL action (e.g., $m_a(next_{HL}) = startHLAction; next_{LL}; endHLAction$). In this way Constraint 3 is satisfied by construction. We also use an additional HL fluent for each action, namely $doneNext$ and $doneUpdate$, to indicate the last action executed. The axioms for these fluents are straightforward. The same can be done for the LL to make refinements consistent.

Finally, we can consider the HL LTL trace constraint:

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

It specifies that moving repeatedly to the next node of the list eventually leads to the last one.

Now, we can prove the following propositions:

Proposition 11. *NDBAT refinement mapping m^{sll} is proper wrt $\mathcal{D}_l^{sll}$.*

Proposition 12. *Let M_h^{sll} be a model of $\mathcal{D}_h^{sll}$ and Ψ the set of trace constraints. $(\mathcal{D}_h^{sll}, M_h^{sll}, \Psi)$ is a temporally lifted abstraction of $\mathcal{D}_l^{sll}$ wrt m^{sll}.*

Strategic Reasoning over Abstractions

Now we want to address the problem of synthesis in the context of temporally lifted abstractions, that is, generating strategies that achieve given goals at the abstract and concrete levels. These strategies are the solutions for the GP problem defined by the abstraction.

For NDBATs, a strong plan is a strategy for the agent that guarantees the achievement of a goal no matter how the environment reacts. (DL21) formalize this notion for state goals and finite traces. They define a strategy as a function from situations to agent actions, i.e. $f(s) = A(\vec{x})$ (note that the value may depend on the entire history). The special agent action $stop$ (with no effects and preconditions) may be returned when the strategy wants to stop (for a finite strategy).

Here, we extend their definition to handle LTL goals and LTL trace constraints over infinite paths. We define $AgtCanForceByIf(Goal, Cstr, f, s)$, meaning that the agent can force a LTL $Goal$ to hold no matter how the environment responds to her actions by following strategy f in situation s if we assume that the LTL trace/path constraint $Cstr$ holds, as follows:

$$\begin{aligned}&AgtCanForceByIf(Goal, Cstr, f, s) \doteq\\ &\quad \forall p.Out(p, f, s) \wedge Holds(Cstr, p) \supset Holds(Goal, p)\end{aligned}$$

where

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

$Out(p, f, s)$ means that path p is a possible outcome of the agent executing strategy f in situation s. Note that if we have a finite set of constraints, we can simply consider $Cstr$ as the conjunction of them. We also define:

$$\begin{aligned}&AgtCanForceIf(Goal, Cstr, s) \doteq\\ &\quad \exists f.AgtCanForceByIf(Goal, Cstr, f, s)\end{aligned}$$

Since in (Khan and Lespérance 2016) paths are defined as infinite sequences of executable situations, we should also require that the strategy is certainly executable, i.e., never prescribes an action that is not executable. It is possible to capture this by defining co-inductively a predicate $CertainlyExecutable(f, s)$, meaning that strategy f is certainly executable in situation s, as follows:

$$\begin{aligned}&CertainlyExecutable(f, s) \doteq\\ &\quad \exists P.[\forall s.P(s) \supset ...] \wedge P(s)\end{aligned}$$

where ... stands for

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

To include this requirement, we would write:

$$\begin{aligned}&AgtCanForceByIf(Goal, Cstr, f, s) \doteq\\ &\quad CertainlyExecutable(f, s) \wedge \forall p.Out(p, f, s) \wedge\\ &\quad Holds(Cstr, p) \supset Holds(Goal, p)\end{aligned}$$

We also need to consider whether the agent is able to execute a program to completion, in particular the implementation of a HL action, no matter how the environment reacts. For this, (DL21) introduce $AgtCanForceBy(\delta, s, f)$, meaning that the agent can ensure that it executes program δ to completion by following strategy f:

$$AgtCanForceBy(\delta, f, s) \doteq \forall P.[\ldots \supset P(\delta, s)]$$

where ... stands for

$$\begin{aligned}&[(f(s) = stop \wedge Final(\delta, s)) \supset P(\delta, s)] \wedge\\ &[\exists A.\exists\vec{t}.(f(s) = A(\vec{t}) \neq stop \wedge\\ &\exists e.\exists\delta'.Trans(\delta, s, \delta', do(A(\vec{t}, e), s)) \wedge\\ &\forall e.(\exists\delta'.Trans(\delta, s, \delta', do(A(\vec{t}, e), s))) \supset \exists\delta'.\\ &Trans(\delta, s, \delta', do(A(\vec{t}, e), s)) \wedge P(\delta', do(A(\vec{t}, e), s))\\ &\supset P(\delta, s)]\end{aligned}$$

Now we can talk about planning with abstractions and how a plan at the abstract level is related to one at the concrete level. As in (BDL23), we impose an additional constraint on action implementation which requires that for any HL agent action that is executable at the LL, the agent has a strategy to execute it no matter how the environment reacts:

Constraint 13 (Agent Can Always Execute HL actions)**.** *For every HL action A, there exists a LL strategy f_A such that for every HL system action sequence $\vec{\alpha}$:*

$$\begin{aligned}\mathcal{D}_l \models \forall s.Do(m(\vec{\alpha}), S_0, s) \supset \\ (\forall \vec{x}.\exists s'.Do_{ag}(m_a(A(\vec{x})), s, s') \supset \\ AgtCanForceBy(m_a(A(\vec{x})), f_A, s))\end{aligned}$$

We can also show that if a LTL formula ψ holds on a HL path p_h, then the mapped version of ψ must hold at the LL on refinements of p_h in m-similar models. First, we define:

Definition 14. *A LL path p_l in M_l* is a refinement of *a HL path p_h in M_h wrt mapping m iff for every finite HL system action sequence $\vec{a_h}$*

if $M_h \models \exists s.OnPath(p_h, s) \wedge Do(\vec{a_h}, S_0, s)$
then $M_l \models \exists s.OnPath(p_l, s) \wedge Do(m_s(\vec{a_h}), S_0, s)$

This means that p_l is a refinement of p_h if it contains a refinement of the infinite action sequence that occurs over p_h. Then we can show that:

Lemma 15. *Consider $M_h \sim_m^{\leftarrow} M_l$ for which Constraint 3 holds and an LTL formula ψ. If $M_h \models Holds(\psi, p_h)$ and p_l in M_l is a refinement of p_h in M_h wrt m, then $M_l \models Holds(m_t(\psi), p_l)$.*

Proof. Consider a given HL system action sequence $\vec{a_h}$. Since $M_h \sim_m^{\leftarrow} M_l$, we know that every time an action from $\vec{a_h}$ is completed both at the HL and (refined) at the LL, the paths p_h and p_l have m-similar situations. Since $M_h \models Holds(\psi, p_h)$, we can show that $M_l \models Holds(m(\psi), p_l)$, i.e. the refinement of ψ holds in path p_l, by induction on the structure of ψ using Definition 1. $\square$

Then, we can prove our main result, that is, given a temporally lifted abstraction, if the agent has a strategy to achieve a LTL goal assuming some LTL constraints at the high level, then there exists a refinement of the HL strategy that ensures it achieves the refinement of the goal at the low level:

Theorem 16. *Let $(\mathcal{D}_h, M_h, Cstr)$ be a temporally lifted abstraction of $\mathcal{D}_l$ wrt refinement mapping m s.t. Constraints 3 and 13 hold, and Goal be an LTL goal. Then we have that:*

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

Proof. Since $M_h \models AgtCanForceIf(Goal, Cstr, S_0)$, it follows that there is a HL strategy f_h such that $M_h \models AgtCanForceByIf(Goal, Cstr, f_h, S_0)$. We have that $(\mathcal{D}_h, M_h, Cstr)$ is a temporally lifted abstraction of $\mathcal{D}_l$ wrt refinement mapping m. Let M_l be a model of $\mathcal{D}_l$. By the definition of temporally lifted abstraction, $M_h \sim_m^{\leftarrow} M_l$.

By Constraint 13, for every HL action A, there exists a LL strategy f_A such that for every HL system action sequence $\vec{\alpha}$, we have $\mathcal{D}_l \cup \mathcal{C} \models \forall s.Do(m_a(\vec{\alpha}), S_0, s) \supset (\forall \vec{x}.\exists s'.Do_{ag}(m_a(A(\vec{x})), s, s') \supset AgtCanForceBy(m_a(A(\vec{x})), f_A, s))$.

Given the HL strategy f_h, we can define a corresponding LL strategy f_l as follows: $f_l(s_l) = f_{f_h(s_h)}(s_l)$ where $s_h \sim_m^{M_h, M_l, \leftarrow} s'_l$, $s'_l \leq s_l$ and for every s''_l such that $s'_l < s''_l \leq s_l$, it is not the case that $s_h \sim_m^{M_h, M_l, \leftarrow} s''_l$ (s_h is the HL situation that is similar to the latest predecessor of s_l that has such a similar situation at the HL).

Since $M_h \models AgtCanForceByIf(Goal, Cstr, f_h, S_0)$, every path p_h produced by f_h in M_h that satisfies $Cstr$ also satisfies $Goal$. Since $M_h \sim_m^{\leftarrow} M_l$, for any path p_l such that $M_l \models Out(p_l, f_l, S_0)$, p_l is a refinement of a path p_h in M_h, i.e. it contains a refinement of the infinite action sequence that occurs over p_h. Then the thesis follows by the fact that $Holds(Cstr, p) \supset Holds(Goal, p)) \equiv Holds(\neg Cstr \vee Goal, p)$ using Lemma 15. $\square$

Let $Goal_{LL}$ be an additional LL LTL goal. We say that a strategy f_l is a solution with respect to $Goal_{LL}$ if $\mathcal{D}_l \models AgtCanForceByIf(Goal_{LL}, True, f_l, S_0)$. It is possible to show that the LL strategy f_l obtained by Theorem 16 is a solution for a GP problem if

$$\begin{aligned}\mathcal{D}_l \models \forall p_l.Starts(p_l, S_0) \supset \\ [Holds(m_t(Goal_{HL}), p_l) \supset Holds(Goal_{LL}, p_l)]\end{aligned}$$

Example Cont. Continuing with our running example, let's discuss the HL and LL strategies for finding the minimum value in a list. First, we can prove that:

Proposition 17. *NDBAT $\mathcal{D}_h^{sll}$ and $\mathcal{D}_l^{sll}$ and mapping m^{sll} satisfy constraint 13.*

The HL LTL goal constraints that must be satisfied are:

$$\begin{aligned}&\Diamond\Box\neg hasNext \\ &\Box(lowerThan \leftrightarrow \bigcirc doneUpdate)\end{aligned}$$

The first says that the list must be scanned till the end, while the second says that the value of the register must be updated iff the pointed node has a lower value than the register.

A HL strategy that guarantees satisfying these goals is:

$$f_h(s) = \begin{cases} update_{HL} & \text{if } lowerThan \\ next_{HL} & \text{if } \neg lowerThan \wedge hasNext \\ stop & \text{otherwise} \end{cases}$$

This strategy prescribes updating the value of the register whenever the node has a lower value and moving the cursor when it is not at the end of the list. As stated before, since we have a propositional HL specification, we can write it in LTL and rely on LTL synthesis engines to automatically derive this strategy (this requires translating the HL specification to LTL). Figure 1 shows the controller obtained by using the engine Strix (Meyer, Sickert, and Luttenberger 2018), as done by (B20). It is consistent with f_h.

The strategy at the LL can be refined as follows:

$$f_l(s) = \begin{cases} update_{LL} & \text{if } cmp = \text{LT} \\ next_{LL} & \text{if } cmp = \text{GEQ} \wedge pos < length \\ no\_op & \text{otherwise} \end{cases}$$

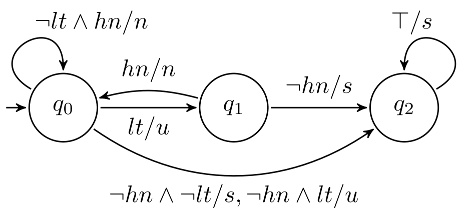

Figure 1: Controller for finding the minimum in a list.

We can handle more complex data structure programming problems (graph traversal, binary tree traversal, sorting of arrays, ...) and GP problems (maze solving, swamp crossing, ...) in a similar way; see the extended version for additional detailed examples. As shown in (B20), this approach scales well in terms of this level of complexity, as the LTL formulas needed for these problems remains simple and compact.

Let M_h be a model of $\mathcal{D}_h^{sll}$, $Goal$ the conjunction of HL LTL goal constraints and $Cstr$ the conjunction of HL LTL trace constraints. We can use Theorem 16 to show that:

$$M_h \models AgtCanForceByIf(Goal, Cstr, f_h, S_0)$$
$$\text{and } \mathcal{D}_l \models AgtCanForceBy(m_t(Goal), True, f_l, S_0)$$

At the LL, a $Goal_{LL}$ can be specified to require that the list does not change and that the register contains the minimum value. It is easy to see that f_l is a solution for $Goal_{LL}$ and for every LL path, if $m_t(Goal)$ holds then $Goal_{LL}$ also holds.

Discussion

In this paper, we have presented a synthesis-based framework to solve a GP problem by using the nondeterministic situation calculus and LTL synthesis. Our methodology involves the following main steps:

1. *Formalize the concrete planning problem instances in the situation calculus* - this amounts to writing a specification for the domain of interest, and is straightforward.
2. *Specify a propositional temporally lifted abstraction as a HL NDBAT* - this abstracts over some details and includes nondeterministic actions; some LTL trace constraints will also be introduced to capture restrictions on the possible histories; obtaining the HL NDBAT is similar to the previous step and, in many cases, we can reuse (part of) the specification of one GP task for other similar tasks (i.e., involving the same data structure).
3. *Write the LTL goals* - this step is application-dependent.
4. *Run a LTL synthesis engine on the HL abstraction* - this automatically derives a HL strategy to reach the goals; notice that our HL propositional abstraction can always be interpreted as an LTL specification.
5. *Translate the HL strategy to a LL program* - this step can be handled simply by using the refinement mapping.

This methodology yields provably correct solutions with strong formal guarantees. Our framework requires the modeler to provide the specifications and the mapping. However, there is no need to generate the entire situation calculus and LTL specifications from scratch. Instead, one could build a library of specifications and reuse them in a modular way. This means that the modeler can just specify her problem in terms of HL trace and goal constraints, exploiting this library, and then run the automatic synthesis engine.

Our work provides a general-purpose framework for GP problems. We have illustrated its use in the context of program synthesis, since it has long been a dream application that could revolutionize software development. As mentioned earlier, (B20) also addresses program synthesis, but our work goes significantly beyond theirs. They assume that every planning instance in a GP problem is a finite-state deterministic classical planning problem and that the set of actions at the abstract and concrete levels are the same. But how the set of planning instances in a GP problem is specified and how one proves that the temporal assumptions and goal constraints are sound is left open. Furthermore, they focus only on tasks involving data structures. In our situation calculus-based framework, the GP problem is specified formally by a basic action theory and the planning instances are models of this theory, which need not be finite-state and can refer to data. The abstract actions can be implemented by programs at the concrete level and we notably ensure the correctness of synthesized HL programs for the LL. One can use situation calculus reasoning techniques to show that the LTL trace assumptions and goals in our temporally lifted abstractions are satisfied. We also provide a way to prove that one has a temporally lifted abstraction.

Our approach also differs from (BDL23) as we address GP tasks while also handling LTL constraints. At the same time, we are less demanding as we rely on simulation rather than bisimulation. Furthermore, we show a way to exploit LTL synthesis engines starting from situation calculus theories. It is worth noting that the GP problem in a first-order (FO) setting is highly general and inherently undecidable, being Turing complete (Lin and Levesque 1998). Nevertheless, we provide a methodology to generate provably correct plans. In contrast, traditional synthesis in propositional planning is known to be decidable and, for finite traces, also scalable à la model checking. However, these results do not generalize easily to a FO state setting.

The work of (Cui, Liu, and Luo 2021; Cui, Kuang, and Liu 2023) is also related to ours. They use the situation calculus with FO state and they address GP, but their approach to the problem achieves only partial correctness, since their notion of a strong solution guarantees achieving the goal only if the plan does not block. This is a significant limitation. Instead, we handle full LTL specifications, including termination properties guaranteeing total correctness. Additionally, our formulation is simpler and more intuitive, as they do not use the nondeterministic situation calculus and require both m-simulation and m-back-simulation between models of the theories. Finally, their work focuses on QNP abstractions, which limits its applicability.

Acknowledgments

This work has been partially supported by the ERC Advanced Grant WhiteMech (No. 834228), the PRIN project RIPER (No. 20203FFYLK), the PNRR MUR project FAIR (No. PE0000013), the Italian National Ph.D. on Artificial

Intelligence at Sapienza University of Rome, the National Science and Engineering Research Council of Canada, and York University.

References

Abadi, M.; Lamport, L.; and Wolper, P. 1989. Realizable and Unrealizable Specifications of Reactive Systems. In *ICALP*, volume 372 of *Lecture Notes in Computer Science*, 1–17. Springer.

Aminof, B.; De Giacomo, G.; Murano, A.; and Rubin, S. 2019. Planning under LTL Environment Specifications. In *ICAPS*, 31–39.

Banihashemi, B.; De Giacomo, G.; 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.

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, 19th-25th August 2023, Macao, SAR, China*, 3112–3122.

Belle, V.; and Levesque, H. J. 2016. Foundations for Generalized Planning in Unbounded Stochastic Domains. In *KR*, 380–389.

Bonet, B.; De Giacomo, G.; Geffner, H.; Patrizi, F.; and Rubin, S. 2020. High-level programming via generalized planning and LTL synthesis. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 17, 152–161.

Bonet, B.; De Giacomo, G.; Geffner, H.; and Rubin, S. 2017. Generalized planning: non-deterministic abstractions and trajectory constraints. In *IJCAI*, 873–879.

Bonet, B.; and Geffner, H. 2015. Policies that Generalize: Solving Many Planning Problems with the Same Policy. In *IJCAI*, volume 15, 2798–2804.

Church, A. 1963. Logic, arithmetics, and automata. In *Proc. Int. Congress of Mathematicians, 1962*.

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*, 3149–3157.

Cui, Z.; Liu, Y.; and Luo, K. 2021. A Uniform Abstraction Framework for Generalized Planning. In *IJCAI*, 1837–1844.

De Giacomo, G.; and Lespérance, Y. 2021. The Nondeterministic Situation Calculus. In Bienvenu, M.; Lakemeyer, G.; and Erdem, E., eds., *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Online event, November 3-12, 2021*, 216–226.

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.

De Giacomo, G.; Lespérance, Y.; and Muise, C. J. 2012. On supervising agents in situation-determined ConGolog. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes)*, 1031–1038. IFAAMAS.

De Giacomo, G.; Lespérance, Y.; and Pearce, A. R. 2010. Situation Calculus Based Programs for Representing and Reasoning about Game Structures. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010*. AAAI Press.

De Giacomo, G.; Murano, A.; Rubin, S.; Di Stasio, A.; et al. 2016. Imperfect-Information Games and Generalized Planning. In *IJCAI*, 1037–1043.

Green, C. C. 1969. Application of Theorem Proving to Problem Solving. In *IJCAI*, 219–240.

Hu, Y.; and De Giacomo, G. 2011. Generalized planning: Synthesizing plans that work for multiple environments. In *IJCAI*, 918–923.

Hu, Y.; and Levesque, H. J. 2010. A correctness result for reasoning about one-dimensional planning problems. In *Twelfth International Conference on the Principles of Knowledge Representation and Reasoning*.

Khan, S. M.; and Lespérance, Y. 2016. Infinite Paths in the Situation Calculus: Axiomatization and Properties. In Baral, C.; Delgrande, J. P.; and Wolter, F., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*, 565–568.

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

Lin, F.; and Levesque, H. J. 1998. What robots can do: robot programs and effective achievability. *Artificial Intelligence*, 101(1-2): 201–226.

McCarthy, J.; and Hayes, P. J. 1969. Some Philosophical Problems From the Standpoint of Artificial Intelligence. *Machine Intelligence*, 4: 463–502.

Meyer, P. J.; Sickert, S.; and Luttenberger, M. 2018. Strix: Explicit Reactive Synthesis Strikes Back! In *CAV (1)*, volume 10981 of *Lecture Notes in Computer Science*, 578–586. Springer.

Milner, R. 1971. An Algebraic Definition of Simulation Between Programs. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence. London, UK, September 1-3, 1971*, 481–489. William Kaufmann.

Milner, R. 1989. *Communication and concurrency*. PHI Series in computer science. Prentice Hall. ISBN 978-0-13-115007-2.

Pnueli, A. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, 46–57. ieee.

Pnueli, A.; and Rosner, R. 1989. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 179–190.

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

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2008. Learning generalized plans using abstract counting. In *AAAI*, 991–997.

Waldinger, R. J.; and Lee, R. C. T. 1969. PROW: A Step Toward Automatic Program Writing. In *IJCAI*, 241–252.