Situation Calculus Temporally Lifted Abstractions for Generalized Planning – Extended Abstract

Giuseppe De Giacomo[1,3], Yves Lespérance[2] and Matteo Mancanelli[3,*]

[1]University of Oxford, Oxford, UK
[2]York University, Toronto, ON, Canada
[3]Sapienza University, Rome, Italy

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.

Keywords

Generalized Planning, Situation Calculus, Nondeterministic Domains, Abstractions, Strategy Synthesis

1. Overview

In generalized planning (GP), one tries to generate a typically iterative policy that solves an infinite set of similar planning problem instances [1, 2, 3]. 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 generalized planning involve constructing an abstraction and finding a solution for this abstraction which handles all the actual problem instances [4, 5]. We propose a new formal framework for generalized planning based on the situation calculus [6, 7] that allows one to provide an abstract description of the domain and associated LTL trace constraints [5, 8], 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* [9] (DL21) where each agent action $A(\vec{x})$ is accompanied by an environment reaction e outside the agent's control that determines the action's outcome, e.g., a flipped coin may fall head or tail. A nondeterministic basic action theory (NDBAT) can be seen as a special kind of action theory, where we have *system* actions $A(\vec{x}, e)$, successor state axioms $\mathcal{D}_{ssa}$, describing how predicates and functions change after system actions are performed, and action precondition axioms $\mathcal{D}_{poss}$, stating when each system action can occur. [10] (BDL23) have proposed an account of abstraction for NDBATs. They relate a high-level NDBAT to a low-level NDBAT through a *refinement mapping* that specifies how a high-level action is implemented at the low level by a ConGolog program [11, 12]. They then define notions of sound and/or complete abstraction for such NDBATs in terms of a notion of bisimulation between their models. [13, 14] have adapted and extended this kind of approach to solve GP problems, focusing on QNP abstractions.

Workshop on Symbolic and Neuro-Symbolic Architectures for Intelligent Robotics Technology (SYNERGY) co-located with the 21st International Conference on Principles of Knowledge Representation and Reasoning (KR2024), November 2–8, 2024, Hanoi, Vietnam.

*Corresponding author.
giuseppe.degiacomo@cs.ox.ac.uk (G. D. Giacomo); lesperan@eecs.yorku.ca (Y. Lespérance); mancanelli@diag.uniroma1.it (M. Mancanelli)

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. We extend the HL theory with LTL trace constraints to impose fairness assumptions on the possible sequences of nondeterministic actions, e.g., ensuring that if we keep advancing we will eventually reach the list's end. To do this, we leverage the axiomatization of infinite paths introduced by [15]. 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 constraint ψ, we define $Holds(\psi, p)$ (meaning that ψ holds on path p)[1]:

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

Finally, we define a notion of *temporally lifted abstraction*, where every LL trace that is a refinement of a sequence of HL actions is m-similar to 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 that achieves 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 [16] (B20), which proposed an approach for solving program synthesis tasks [17, 18, 19, 20, 21] that involve the manipulation of data structures such as lists, trees, and graphs by viewing them as instances of generalized planning. 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.

2. Methodology

Our methodology involves the following main steps:

1. *Formalize the concrete planning problem instances in situation calculus* - this consists of writing the specification for the domain of interest; hence, it 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 future 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 domain-dependent
4. *Run a LTL synthesis engine on the HL abstraction* - this automatically derives a HL strategy to reach the goals; note that our HL propositional abstraction can always be interpreted as an LTL specification

[1] ϕ is a ground situation-suppressed formula defined over an NDBAT; $\phi[s]$ is the formula obtained by restoring the situation s.

5. *Translate the HL strategy to a LL program* - this step can be simply addressed by using the refinement mapping

This methodology yields provably correct solutions; the following sections will be devoted to providing strong formal guarantees. Note that there should be no need to generate the entire situation calculus 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 engines.

Example: Finding the minimum value in a linked list.

Step 1: Consider the task of finding the minimum value stored in a singly-linked list. We can construct a NDBAT $\mathcal{D}_l^{sll}$ with two actions and two functional fluents. 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 precondition and no effect. Note that, at the LL, we have only one possible environment reaction *Success* for each action. The functional 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.

Step 2: NDBAT $\mathcal{D}_h^{sll}$ models the HL action theory with two (propositional) fluents: $hasNext(situation)$, signaling whether the cursor points at the last node of the list, and $lowerThan(situation)$, signaling whether the value of the node pointed by the cursor is lower than the value stored in the register. The actions used for our task will be $next_{HL}$, which moves the cursor if possible and performs the comparison between the node and register values, and $update_{HL}$, which updates the register's value to the node's. The environment reaction of $next_{HL}$ tells us if the successor node's value is lower than the register's and if the end is reached. Here is a fragment of the refinement mapping m^{sll}:

$m_s(next_{HL}(r_h)) =$
  $next_{LL}(Success_{LLN})$;
  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

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 $doneNext$ and $doneUpdate$, 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:

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

Step 3: The HL LTL goals that must be satisfied are:

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

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.

Step 4: A HL strategy that satisfies the goals is:

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

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 needs an intermediate translation from the HL specification to an LTL one). Figure 1 shows the controller obtained by using the engine Strix [22], as done by (B20). It is easy to see that f_h is consistent with the controller.

Step 5: At the LL the strategy can be refined as follows:

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

3. Temporally Lifted Abstractions

As mentioned previously, we want to specify a GP problem at the concrete level by a BAT $\mathcal{D}_l$, with the various basic planning problems being models of this BAT. We will then define an abstract version of the theory by providing a propositional NDBAT $\mathcal{D}_h$, for which we have a single model and incomplete information, with the nondeterminism capturing the differences between the different concrete instances. We will also specify some LTL trace constraints on the abstract model that characterize the traces that can actually arise in the concrete theory instances. We refer to this framework 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. So we will specify the relationship between the HL NDBAT and the LL BAT by a refinement mapping m. 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 *(i)* m_a associates each HL agent action A to a ConGolog agent program (i.e., $m_a(A(\vec{x})) = \delta_A^{ag}(\vec{x})$), *(ii)* m_s associates each system action A to a ConGolog system program (i.e., $m_s(A(\vec{x}, e)) = \delta_A^{sys}(\vec{x}, e)$), *(iii)* m_f maps each situation-suppressed HL fluent $F(\vec{x})$ to a formula ϕ_F.

Our revisited definition of NDBAT mapping maintains the previous elements and includes a new component m_t, which specifies how HL trace constraints are mapped to the LL:

Definition 1 (Refinement Mapping for Trace Constraints)**.** *Let ψ be an LTL trace constraint and Hlc a distinguished symbol that 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:*

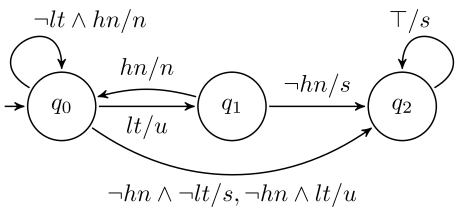

Figure 1: Controller for finding the minimum in a list.

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

As in [10] and [23], respectively, we require the following two constraints to hold:

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

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

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 $\mathcal{D}_l \cup C \models Do(m(\vec{\alpha}), S_0, s)$.*

Temporally lifted abstractions. Finally, we can present the concept of temporally lifted abstractions. To relate the HL and LL models, we first need to present the notion of m-isomorphic situations.

Definition 4 (m-isomorphic situations)**.** *We say that situation s_h in M_h is m-isomorphic to situation s_l in M_l, written $s_h \simeq_m^{M_h,M_l} s_l$, if and only if*

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

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

(BDL23) define a variant of bisimulation [24, 25] to establish a relation based on the refinement mapping. Here, we stick to a unidirectional version:

Definition 5 (m-simulation)**.** *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 $M_l, v[s/s_l, s'/s_l'] \models Do(m(A(\vec{x})), s, s')$, 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 $\langle s_h', s_l'\rangle \in R$*

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 $\langle S_0^{M_h}, S_0^{M_l}\rangle \in R$.

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 both on some path at the HL and on all paths at the LL.

Definition 6 (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 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
  $D_l \models \forall p_l.Starts(p_l, S_{0_l}) \supset Holds(m_t(\psi), p_l)$

It is possible to verify that one has a temporally lifted abstraction by using the following theorem.

Theorem 7. *Suppose that we have a HL NDBAT $\mathcal{D}_h$, a model M_h of $\mathcal{D}_h$, a set of HL LTL state 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 $\phi \in D^h_{S_0}$,*

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

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

where $\phi^{Poss}_{A_i}(\vec{x})$ is the right-hand side (RHS) of the precondition axiom for action $A_i(\vec{x})$,

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

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

where $\phi^{ssa}_{F_i,A_i}(\vec{y}, \vec{x})$ is the RHS of the successor state axiom for F_i instantiated with action $A_i(\vec{x})$ where action terms have been eliminated using $\mathcal{D}^h_{ca}$, and

(e) *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
$D_l \models \forall p_l.Starts(p_l, S_{0_l}) \supset Holds(m_t(\psi), p_l)$,

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

4. 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 at the concrete level. 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:

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

where

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

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

$CertainlyExecutable(f, s)$ 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 $AgtCanForceIf(Goal, Cstr, s) \doteq \exists f.AgtCanForceByIf(Goal, Cstr, f, s)$.

We also need to consider whether the agent is able to execute a program to completion, i.e., 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:

$$
\begin{array}{l}
AgtCanForceBy(\delta, f, s) \doteq \forall P.[\ldots \supset P(\delta, s)] \\
\text{where } \ldots \text{stands for} \\
[(f(s) = stop \wedge Final(\delta, s)) \supset P(\delta, s)] \wedge \\
[\exists A.\exists \vec{t}.(f(s) = A(\vec{t}) \neq stop \wedge \\
\exists e.\exists \delta'.Trans(\delta, s, \delta', do(A(\vec{t}, e), s)) \wedge \\
\forall e.(\exists \delta'.Trans(\delta, s, \delta', do(A(\vec{t}, e), s))) \supset \\
\exists \delta'.Trans(\delta, s, \delta', do(A(\vec{t}, e), s)) \wedge P(\delta', do(A(\vec{t}, e), s)) \\
\supset P(\delta, s)]
\end{array}
$$

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 8 (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 $\vec{\alpha}$:*

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

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 9. *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 8 hold, and $Goal$ be an LTL goal. Then we have that:*

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

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 easy to see that the LL strategy f_l obtained by Theorem 9 is a solution for a GP problem if

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

Acknowledgments

This work has been partially supported by the ERC Advanced Grant WhiteMech (No. 834228), 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

[1] S. Srivastava, N. Immerman, S. Zilberstein, Learning generalized plans using abstract counting, in: AAAI, 2008, pp. 991–997.

[2] Y. Hu, G. De Giacomo, Generalized planning: Synthesizing plans that work for multiple environments, in: IJCAI, 2011, pp. 918–923.

[3] V. Belle, H. J. Levesque, Foundations for generalized planning in unbounded stochastic domains, in: KR, 2016, pp. 380–389.

[4] B. Bonet, H. Geffner, Policies that generalize: Solving many planning problems with the same policy., in: IJCAI, volume 15, 2015, pp. 2798–2804.

[5] B. Bonet, G. De Giacomo, H. Geffner, S. Rubin, Generalized planning: non-deterministic abstractions and trajectory constraints, in: IJCAI, 2017, pp. 873–879.

[6] J. McCarthy, P. J. Hayes, Some Philosophical Problems From the Standpoint of Artificial Intelligence, Machine Intelligence 4 (1969) 463–502.

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

[8] B. Aminof, G. De Giacomo, A. Murano, S. Rubin, Planning under LTL environment specifications, in: ICAPS, 2019, pp. 31–39.

[9] G. De Giacomo, Y. Lespérance, The nondeterministic situation calculus, in: M. Bienvenu, G. Lakemeyer, E. Erdem (Eds.), Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Online event, November 3-12, 2021, 2021, pp. 216–226. URL: https://doi.org/10.24963/kr.2021/21. doi:10.24963/KR.2021/21.

[10] B. Banihashemi, G. De Giacomo, Y. Lespérance, 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, 2023, pp. 3112–3122. URL: https://doi.org/10.24963/ijcai.2023/347. doi:10.24963/IJCAI.2023/347.

[11] G. De Giacomo, Y. Lespérance, H. J. Levesque, ConGolog, a concurrent programming language based on the situation calculus, Artificial Intelligence 121 (2000) 109–169. URL: https://doi.org/10.1016/S0004-3702(00)00031-X. doi:10.1016/S0004-3702(00)00031-X.

[12] G. De Giacomo, Y. Lespérance, A. R. Pearce, 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, 2010. URL: http://aaai.org/ocs/index.php/KR/KR2010/paper/view/1388.

[13] Z. Cui, Y. Liu, K. Luo, A uniform abstraction framework for generalized planning., in: IJCAI, 2021, pp. 1837–1844.

[14] Z. Cui, W. Kuang, Y. Liu, Automatic verification for soundness of bounded qnp abstractions for generalized planning, in: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, 2023, pp. 3149–3157.

[15] S. M. Khan, Y. Lespérance, Infinite paths in the situation calculus: Axiomatization and properties, in: C. Baral, J. P. Delgrande, F. Wolter (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016, 2016, pp. 565–568. URL: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12830.

[16] B. Bonet, G. De Giacomo, H. Geffner, F. Patrizi, S. Rubin, High-level programming via generalized planning and ltl synthesis, in: Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning, volume 17, 2020, pp. 152–161.

[17] C. C. Green, Application of theorem proving to problem solving, in: IJCAI, 1969, pp. 219–240.

[18] R. J. Waldinger, R. C. T. Lee, PROW: A step toward automatic program writing, in: IJCAI, 1969, pp. 241–252.

[19] A. Church, Logic, arithmetics, and automata, in: Proc. Int. Congress of Mathematicians, 1963, pp. 23–35.

[20] M. Abadi, L. Lamport, P. Wolper, Realizable and unrealizable specifications of reactive systems, in: ICALP, volume 372 of *Lecture Notes in Computer Science*, Springer, 1989, pp. 1–17.

[21] A. Pnueli, R. Rosner, On the synthesis of a reactive module, in: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, 1989, pp. 179–190.
[22] P. J. Meyer, S. Sickert, M. Luttenberger, Strix: Explicit reactive synthesis strikes back!, in: CAV (1), volume 10981 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 578–586.
[23] Y. Lespérance, G. D. Giacomo, M. Rostamigiv, S. M. Khan, Abstraction of situation calculus concurrent game structures, in: Proceedings of the AAAI Conference on Artificial Intelligence, AAAI 2024, 2024, pp. 10624–10634. URL: https://doi.org/10.1609/aaai.v38i9.28933. doi:10.1609/AAAI.V38I9.28933.
[24] R. Milner, An algebraic definition of simulation between programs, in: Proceedings of the 2nd International Joint Conference on Artificial Intelligence. London, UK, September 1-3, 1971, William Kaufmann, 1971, pp. 481–489. URL: http://ijcai.org/Proceedings/71/Papers/044.pdf.
[25] R. Milner, Communication and concurrency, PHI Series in computer science, Prentice Hall, 1989.