

Situation Calculus Temporally Lifted Abstractions for Program Synthesis

Giuseppe De Giacomo^{a,c}, Yves Lespérance^b, and Matteo Mancanelli^c

^aUniversity of Oxford, (Oxford, UK), ^bYork University (Toronto, ON, Canada), ^cSapienza University (Rome, Italy)



Motivation

Objective: We address the program synthesis task to create concrete programs from high-level (HL) abstractions of data structure behavior.

Tools: We use nondeterministic SitCalc for the specifications, ConGolog programs to map HL into LL, and LTL for trace constraints and goals.

Overview: We consider a single (propositional) HL action theory/model with incomplete information and a concrete LL action theory with several models and complete information. We formally prove that if an agent has a strategy to achieve a goal under LTL constraints at the HL, then there exists a refinement of the strategy at the LL.

Nondeterministic SitCalc, ConGolog and LTL

A Nondeterministic Basic Action Theory (NDBAT) is an extension of classical BATs handling nondeterministic actions. For each **agent action** $A(\vec{x})$, the environment selects a reaction e to produce the **system action** $A(\vec{x}, e)$.

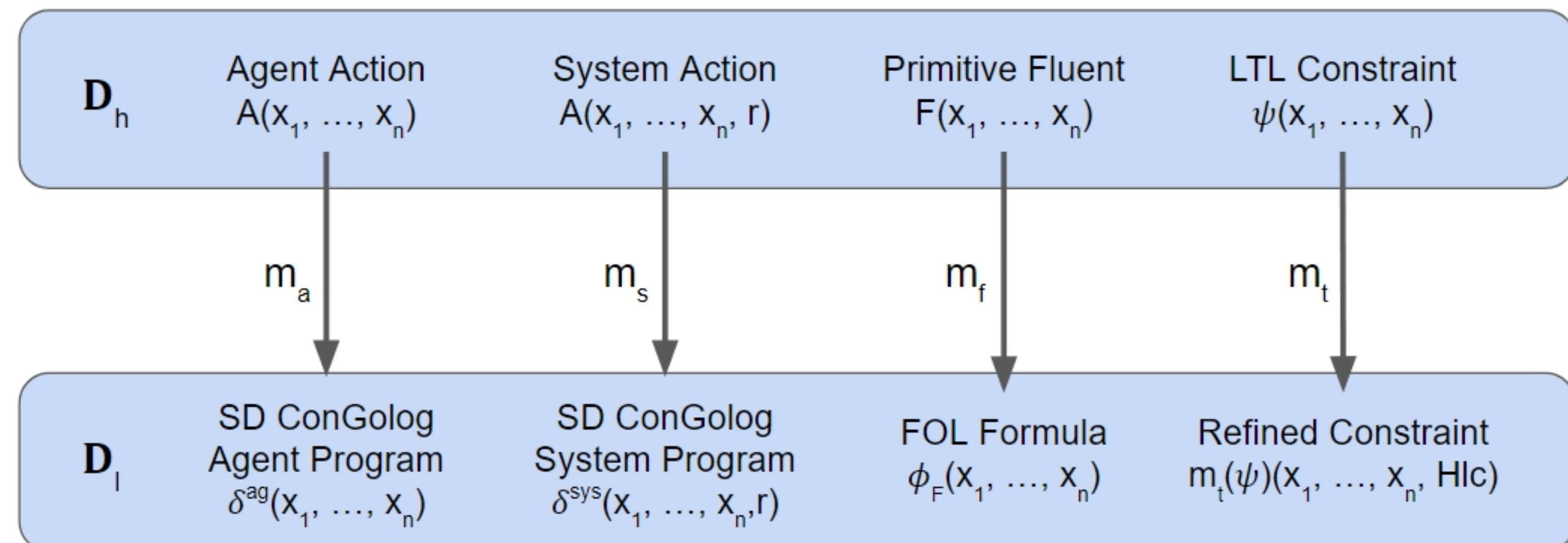
ConGolog is a **HL programming language** that supports complex action sequences, whose constructs include sequential execution, nondeterministic choice, variable binding, iteration, and interleaving.

LTL is a formalism for expressing temporal properties of reactive systems. LTL synthesis involves generating a **controller** that satisfies a LTL goal and it can be exploited in the context of **generalized planning problems**.

We impose LTL trace constraints for **filtering world histories** in NDBATs, leveraging on the axiomatization for infinite paths and the special symbol $Holds(\psi, p)$ (meaning that a constraint ψ holds on a path p).

Refinement Mapping m

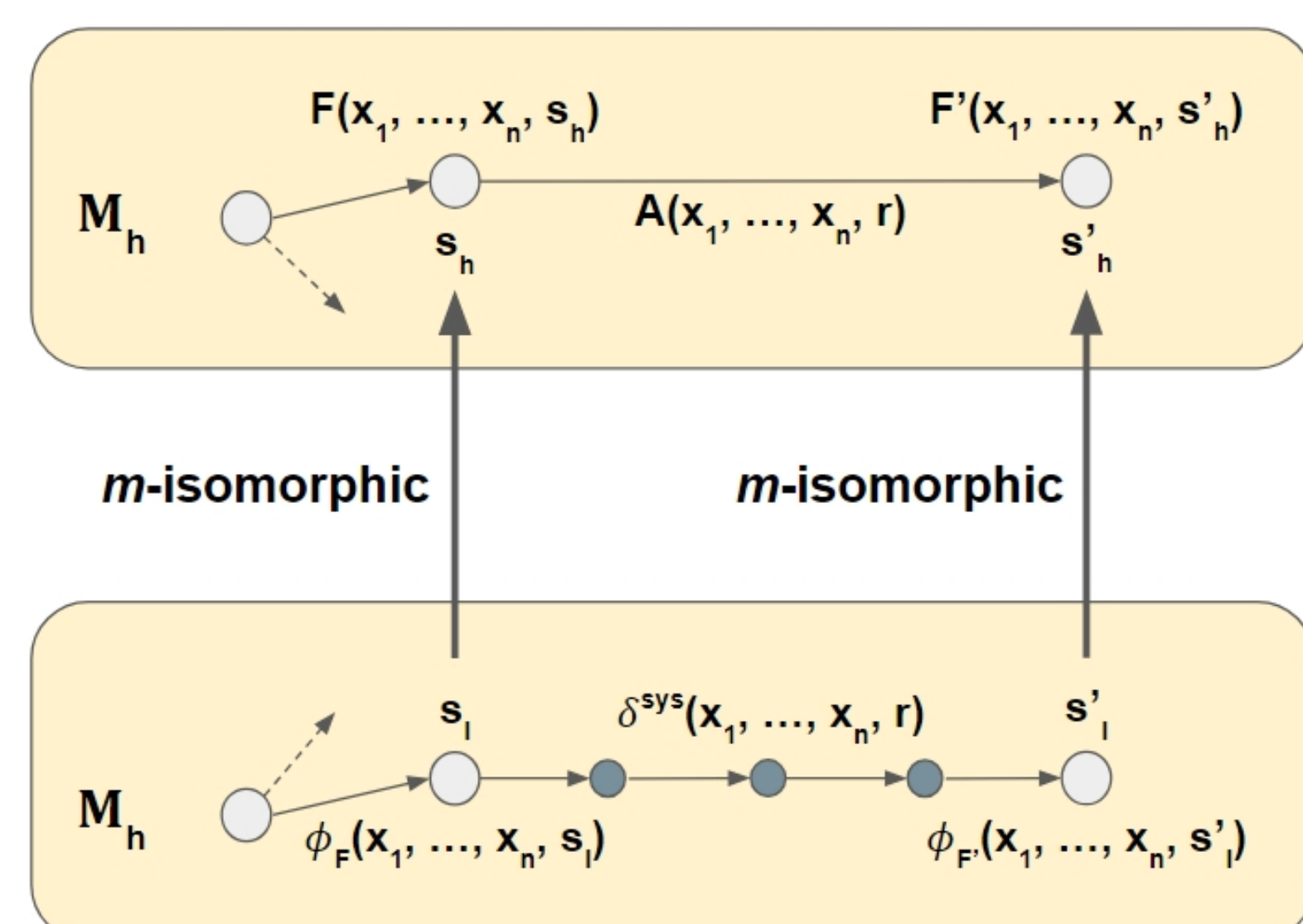
An NDBAT **refinement mapping** m is a tuple $\langle m_a, m_s, m_f, m_t \rangle$. In defining m_t to map HL constraints into LL ones, we suppose that the LL theory tracks when refinements of HL actions end using a state formula $Hlc(s)$, meaning that a HL action has just completed in situation s .



m -Simulation

To relate the HL and LL NDBATs, we revisit the notion of **bisimulation**, sticking to a unidirectional version called **m -simulation**.

Two situations s_h and s_l are **m -isomorphic** iff they evaluate all HL fluents the same.



Two models M_h and M_l are **m -similar** if the initial situations are m -isomorphic and the resulting s'_l after executing $m(A)$ at the LL is always m -isomorphic to the resulting s'_h after executing A at the HL.

Temporally Lifted Abstraction

Definition 1 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 \mathcal{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 $D_l \models \forall p_l. Starts(p_l, S_{0_l}) \supset Holds(m_t(\psi), p_l)$

Example: Minimum in a List

Description: We illustrate our framework addressing the problem of finding the minimum value in a singly-linked list.

LL: We represent a list **deterministically** with predicates identifying the head, each node's value and successor, an iterator and a register. The actions are common methods manipulating the previous elements.

HL: We abstract details using nondeterministic actions *next* (move cursor), *checkValue* (compare values), and *update* (update register). The **environment reaction** of *next* tells if the end is reached, and *checkValue* tells if the current node's value is lower than the registers'.

HL Successor State Axioms:

$hasNext(do(a, s)) \equiv$
 $hasNext(s) \wedge a \neq next(End)$
 $lowerThan(do(a, s)) \equiv$
 $a = checkValue(LT) \vee$
 $lowerThan(s) \wedge$
 $a \neq checkValue(GEQ) \wedge$
 $a \neq update(Success_{HU}) \wedge$
 $\forall r. a \neq next(r)$

Fragment of Refinement Mapping:

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

HL LTL Trace Constraint: $(\Box \Diamond doneNext) \rightarrow \Diamond \neg hasNext$

i.e., moving repeatedly to the next node eventually leads to the last one

Results about Strategic Reasoning

A **strong plan** is a strategy ensuring goal achievement **regardless of environment reactions**. We define $AgtCanForceByIf(Goal, Cstr, f, s)$, meaning that the agent can force a LTL *Goal* by following strategy f in s if LTL path constraint $Cstr$ holds:

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

$CertExec(f, s)$ means that f certainly prescribes executable actions.

$Out(p, f, s)$ means that p is a possible outcome of the agent executing f .

We can prove that **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 2 Let $(\mathcal{D}_h, M_h, Cstr)$ be a temporally lifted abstraction of \mathcal{D}_l wrt refinement mapping m s.t. constraints about actions execution hold^a, and *Goal* be an LTL goal. If $M_h \models AgtCanForceIf(Goal, Cstr, S_0)$, then **there exist a LL strategy f_l** such that $\mathcal{D}_l \models AgtCanForceByIf(m(Goal), True, f_l, S_0)$

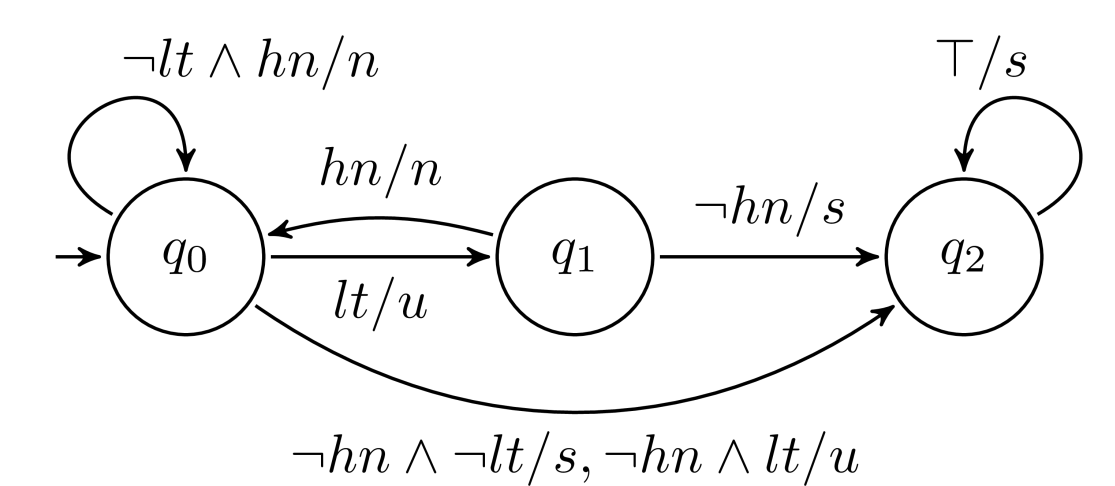
^afor a comprehensive discussion, please refer to the full paper

Example: Synthesize and Refine a Strategy

HL LTL Goal:

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

Controller:



Here is a strategy that **guarantees** the satisfaction of the goals:

$f_h(s) = \begin{cases} checkValue & \text{if } doneNext(s) \\ update & \text{if } lowerThan(s) \\ next & \text{if } \neg lowerThan(s) \wedge hasNext(s) \\ stop & \text{otherwise} \end{cases}$

The previous controller can be **generated automatically** by an LTL synthesis engine like Strix, and it is perfectly consistent with f_h . By Theorem 2, **there must exist** a refinement of f_h at the LL.

Conclusion

Future works:

- Create modular libraries of verified abstractions for common data structures
- Explore partial automation of specification generation

Scan for details and contacts:

