arXiv:2605.12539v1 [cs.LO] 5 May 2026

ocLTL: LTL Realizability and Synthesis Modulo ω-Categorical Structures

Ohad Asor*

Abstract

We introduce ocLTL, a version of LTL+P modulo ω-categorical theories. We reduce its realizability and synthesis problems into the corresponding problems in propositional LTL+P. The core of the reduction replaces each data subformula with a finite disjunction over complete types. The complexity remains 2-EXPTIME with additional blowup that depends only on the theory but not the formula.

1 Introduction

LTL modulo infinite data domain is an active area of research [6, 3, 7, 11, 14, 12, 1]. There have been some decidability [6, 3, 9, 16, 8, 14, 12] and undecidability [3, 7, 15, 2] results. In this paper we focus on realizability and synthesis of LTL modulo ω-categorical structures. In contrast to some works [14, 11, 10], we do allow data variables to interact over time, however in a bounded fashion. Full LTL+Past over infinite traces is supported. The key tool is the Ryll-Nardzewski theorem: a countable structure is ω-categorical iff it has finitely many complete k-types for each k, all of which are isolated.

2 Preliminaries

Fix $\mathcal{M}$ to be a countable ω-categorical structure over a signature Σ and domain M. We denote by T_k the set of all complete k-types in $\mathcal{M}$.

Theorem 1 (Ryll-Nardzewski, Engeler, Svenonius [13])**.** *The following are equivalent:*

1. *$\mathcal{M}$ is ω-categorical.*
2. *For each k, $\mathcal{M}$ has only finitely many complete k-types.*
3. *Each complete k-type in $\mathcal{M}$ contains only finitely many formulas up to logical equivalence (i.e. it is* isolated*).*
4. *Two tuples from $\mathcal{M}$ have the same type, iff they lie in the same* $\mathrm{Aut}(\mathcal{M})$*-orbit.*

*ohad@idni.org

Remark 2. By theorem 1, any formula $\phi(\bar{x})$ with k free variables can be encoded as a finite set of types. Define $\iota(\phi) = \{\tau \in T_k \mid \phi \in \tau\}$. Then for any k-tuple $\bar{a}$, $\mathcal{M} \models \phi(\bar{a})$ if and only if $\mathrm{tp}(\bar{a}) \in \iota(\phi)$. Further, ϕ is logically equivalent to $\bigvee_{\tau \in \iota(\phi)} \tau$.

Definition 3 (Restriction)**.** For $\tau \in T_n$ and an index set $I \subseteq \{1, \ldots, n\}$, the restriction $\tau|_I \in T_{|I|}$ is the complete type obtained by taking only formulas in the type that mention only the variables $\{x_i\}_{i \in I}$, after appropriate renumeration of the indices$x_1, \ldots, x_{|I|}$.

It is easy to see that the restriction gives a complete type indeed, given that the original type is complete.

Proposition 4 (Extension)**.** *For every $\bar{a} \in M^k$ and every $\tau \in T_{k+1}$ s.t. $\mathcal{M} \models \tau|_{\{1,\ldots,k\}}(\bar{a})$ there exists $b \in M$ s.t. $\mathcal{M} \models \tau(\bar{a}, b)$.*

Proof. Since τ is realized, some $(\bar{a}', b')$ has type τ. Then $\mathrm{tp}(\bar{a}') = \tau|_{\{1,\ldots,k\}} = \mathrm{tp}(\bar{a})$, so by theorem 1, some automorphism α maps $\bar{a}'$ to $\bar{a}$. Then $\mathrm{tp}(\bar{a}, \alpha(b')) = \mathrm{tp}(\bar{a}', b') = \tau$. □

Definition 5 (Product structure)**.** For $n > 0$ define the power structure $\mathcal{M}^n$ to have the set of all n-tuples from M, inheriting the same signature extended with the projection functions.

The following is immediate too since Σ is still operating over single elements via the projection functions:

Proposition 6. *If $\mathcal{M}$ is ω-categorical then so is $\mathcal{M}^n$. Moreover, there is a canonical bijection between the k-types of $\mathcal{M}^n$ and the kn-types of $\mathcal{M}$.*

Moreover, this extends to a product of distinct ω-categorical structures.

Finally, we assume $\mathcal{M}$ is *effectively presented*: the complete type of any tuple is computable, and the witness b in proposition 4 can be computed from $\bar{a}$ and τ.

3 ocLTL($\mathcal{M}$)

A reactive system reads inputs and produces outputs, at each time step. Inputs and outputs take values in a data domain M. For simplicity we assume the number of input and output streams are equal, denote it by s. A *program* is a function $F : (M^\omega)^s \to (M^\omega)^s$ that is *causal* , i.e.

$$\forall n \forall a, b \in (M^\omega)^s . \mathrm{pr}_n(a) = \mathrm{pr}_n(b) \to \mathrm{pr}_n(F(a)) = \mathrm{pr}_n(F(b))$$

where pr_n is the prefix function returning the first n elements of a word, and its generalization to tuples of words extends componentwise.

3.1 Syntax and Semantics

Fix a *lookback* parameter $\ell \geq 0$. A *data formula* is a first-order Σ-formula whose free variables are x^i_{-j}, y^i_{-j} for $1 \leq i \leq s$ and $0 \leq j \leq \ell$. The x's are the *inputs* and the y's are the *outputs*. A trace is a function $\pi : \mathbb{N} \to M^s \times M^s$. Denote $\pi_{x^i}(t)$ and $\pi_{y^i}(t)$ for the i'th input and output at time t, respectively.

Remark 7. It is easy to support also lookahead by allowing x^i_{+j}, y^i_{+j} as well, but we keep the lookback-only setting for simplicity.

Definition 8. $\mathsf{ocLTL}(\mathcal{M})$ formulas have the syntax

$$\varphi \coloneqq \delta \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathsf{X}\varphi \mid \varphi_1 \mathsf{U} \varphi_2 \mid \mathsf{Y}\varphi \mid \varphi_1 \mathsf{S} \varphi_2$$

under the semantics

$$\begin{aligned}
\pi, t \geq \ell \models \delta &\iff \mathcal{M} \models \delta \left[x^i_{-j} \mapsto \pi_{x^i}(t-j), y^i_{-j} \mapsto \pi_{y^i}(t-j) \right] \\
\pi, t < \ell \models \delta &\iff \bot \\
\pi, t \models \neg\varphi &\iff \pi, t \not\models \varphi \\
\pi, t \models \varphi_1 \wedge \varphi_2 &\iff \pi, t \models \varphi_1 \wedge \pi, t \models \varphi_2 \\
\pi, t \models \mathsf{X}\varphi &\iff \pi, t+1 \models \varphi \\
\pi, t \models \varphi_1 \mathsf{U} \varphi_2 &\iff \exists k \geq t . \pi, k \models \varphi_2 \wedge \forall t \leq j < k . \pi, j \models \varphi_1 \\
\pi, t \models \mathsf{Y}\varphi &\iff t > 0 \wedge \pi, t-1 \models \varphi \\
\pi, t \models \varphi_1 \mathsf{S} \varphi_2 &\iff \exists k \leq t . \pi, k \models \varphi_2 \wedge \forall k < j \leq t . \pi, j \models \varphi_1
\end{aligned}$$

We write $\mathsf{G}\varphi \coloneqq \neg(\top \mathsf{U} \neg\varphi)$ and $\mathsf{F}\varphi \coloneqq \top \mathsf{U} \varphi$. A program *realizes* φ if every induced trace π satisfies $\pi, 0 \models \varphi$.

Remark 9. Speaking about synthesis alongside model-theoretic types introduces an unfortunate terminological clash. We can speak about a program realizing a specification. But when we say that types are realized in a model, or realized by a tuple of domain elements, we refer to a different concept. The meaning of "realize" will always be clear from the context.

3.2 Three Variables

To avoid cumbersome notation we assume that data subformulas contain each only three free variables: m, x, y (memory, current input, current output). Bringing formulas to this form straightforward: first introduce auxiliary output streams, each of them responsible to carry the values from the bounded past (this can easily be expressed in $\mathsf{ocLTL}(\mathcal{M})$). Then pack everything into a product structure (to collapse multiple input and output streams, including the auxiliary ones, into single input and output streams) and obtain the desired form. It is easy to generalize all derivations in this paper to the general case, one that involves neither the reduction into three variables nor moving to the product structure, but the notation becomes more complex.

4 Realizability

In this section we shall reduce $\mathsf{ocLTL}(\mathcal{M})$ into an equirealizable LTL formula. As concluded in section 3.2, we assume that all data subformulas have m, x, y only as free variables, so in particular $s = \ell = 1$. By theorem 1, we can now write each data subformula as a disjunction of types from T_3 (the Q's below). After doing so, we replace each type with a propositional variable. The inputs in the translated LTL formula will encode types from T_2 (the P's below) representing the current *partial state* (m, x) which is the previous output (the memory) and the current input. This gives rise to an assume-guarantee structure: we assume that the inputs are well-formed in this fashion, e.g. by an encoder sitting between the program and the environment. The outputs will encode the *full state* (m, x, y). We will then encode in LTL how those variables interact: each tuple should realize

exactly one type, the partial state type should conform to the type before it (the assume-guarantee structure), and the full state type should conform to the partial state. This all is formalized as follows:

1. Replace each data subformula δ with $\bigvee_{\tau \in \iota(\delta)} Q_\tau$ and obtain φ^* (cf. 2).

2. Express the uniqueness constraints:

$$\Phi_I \coloneqq \bigvee_{\sigma \in T_2} P_\sigma \wedge \bigwedge_{\sigma_1 \neq \sigma_2} \neg \left(P_{\sigma_1} \wedge P_{\sigma_2}\right)$$
$$\Phi_O \coloneqq \bigvee_{\tau \in T_3} Q_\tau \wedge \bigwedge_{\tau_1 \neq \tau_2} \neg \left(Q_{\tau_1} \wedge Q_{\tau_2}\right)$$

   since each tuple realizes exactly one type.

3. Express the compatibility of the full state type with the partial state type:

$$\Psi_O \coloneqq \bigwedge_{\tau \in T_3} Q_\tau \to P_{\tau|_{m,x}}$$

   (cf. figure 1)

4. Express the compatibility of the partial state type with the previous full state type (written in a future-looking way):

$$\Psi_I \coloneqq \bigwedge_{\tau \in T_3} Q_\tau \to \mathsf{X} \bigvee_{\sigma \in S_\tau} P_\sigma$$

   where

$$S_\tau = \{\sigma : \sigma|_m = \mathsf{sh}\left(\tau\right)|_m\}$$

   and $\mathsf{sh}\left(\tau\right)$ takes $\tau\left(m, x, y\right)$ and "shifts" it: it renames its variables into $\tau\left(u, v, m\right)$. The variables u, v are unused because we immediately restrict to m (cf. figure 1).

5. The translated specification is:

$$\hat{\varphi} \coloneqq \mathsf{G}\left(\Psi_I \wedge \Phi_I\right) \to \left(\varphi^* \wedge \mathsf{G}\left(\Psi_O \wedge \Phi_O\right)\right).$$

Declare the P's as inputs and the Q's as outputs. The result has $|T_2| + |T_3|$ propositional variables.

Theorem 10 (Equirealizability)**.** *φ is realizable over $\mathcal{M}$ iff $\hat{\varphi}$ is realizable.*

Proof. Immediate from the construction: each direction uses proposition 4 and the effective presentation assumption to find concrete witnesses matching a prescribed type, and theorem 1 to transfer truth values between the concrete and propositional traces. We nevertheless present a formal proof:

Assume F realizes φ. We construct a propositional strategy G realizing $\hat{\varphi}$. G only needs to satisfy $\Psi_O \wedge \Phi_O \wedge \varphi^*$ whenever the environment satisfies the $\Psi_I \wedge \Phi_I$. At $t = 0$ let $m_0 \in M$ be arbitrary. For $t > 0$ the environment provides a valuation for the P variables. By Φ_I, exactly one P_σ holds. Ψ_I guarantees that $\sigma_t|_m = \mathsf{sh}\left(\tau_{t-1}\right)|_m$. Since τ_{t-1} was realized by $m_{t-1}, x_{t-1}, y_{t-1}$, setting $m_t = y_{t-1}$ realizes the m-component of σ_t. By 4, there exists a concrete $x_t \in M$ such that $\mathrm{tp}\left(m_t, x_t\right) = \sigma_t$. We feed the history of inputs up to x_t into F which causally produces an output y_t. We then compute the complete type $\tau_t = \mathrm{tp}\left(m_t, x_t, y_t\right)$. G then outputs $Q_\tau = \top$ and $Q_{\tau'} = \bot$

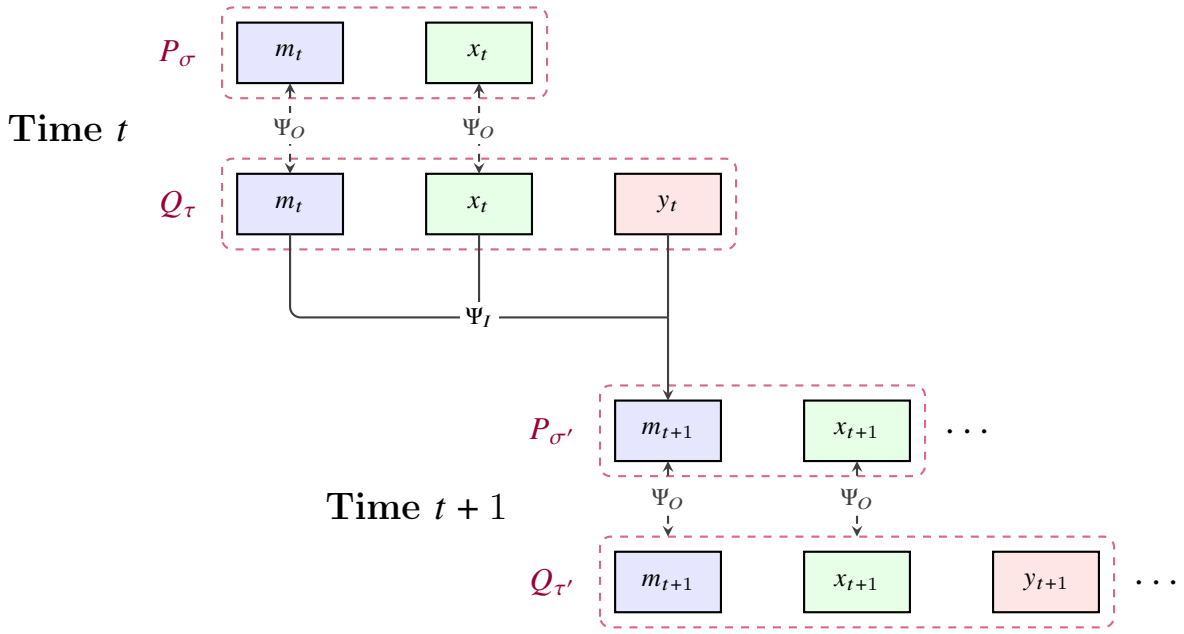


Figure 1: The two compatibility constraints across time steps. At time t, the environment declares a 2-type $P_\sigma = \mathrm{tp}(m_t, x_t)$ and the system responds with a 3-type $Q_\tau = \mathrm{tp}(m_t, x_t, y_t)$. The constraint Ψ_O requires that τ restricts to σ on (m, x): the system's output type must extend the environment's. The constraint Ψ_I links consecutive steps: since y_t becomes the memory m_{t+1}, the 1-type $\mathsf{sh}(\tau)|_m$ determined by the current output must equal the memory component $\sigma'|_m$ of any input type $P_{\sigma'}$ the environment inputs at time $t+1$.

for all $\tau' \neq \tau_t$, so Φ_O holds. Ψ_O holds because $\tau_t|_{m,x} = \mathrm{tp}(m_t, x_t) = \sigma_t$. Since the concrete trace generated by F satisfies φ, and φ^* is derived by replacing every data formula δ with $\bigvee_{\tau \in \iota(\delta)} Q_\tau$, theorem 1 guarantees that the sequence of types τ_t satisfies φ^*.

For the other direction, assume G realizes $\hat{\varphi}$. We construct a program F over $\mathcal{M}$ realizing φ. At $t = 0$, we have an input x_0 and we pick an arbitrary $m_0 \in M$. For $t > 0$, given the memory m_t and the new input x_t, we compute $\sigma_t = \mathrm{tp}(m_t, x_t)$. We feed the environment valuation $P_{\sigma_t} = \top$ (and all others $\bot$) to G. Φ_I is satisfied by construction. We set $m_t = y_{t-1}$ as expected and by that we also ensure that memory part of σ_t matches $\mathsf{sh}(\tau_{t-1})|_m$, satisfying Ψ_I. By assumption, G now outputs a valuation satisfying $\Psi_O \wedge \Phi_O \wedge \varphi^*$. By Φ_O, exactly one variable Q_{τ_t} is true. By Ψ_O, $\tau_t|_{m,x} = \sigma_t$, so τ_t restricts to $\mathrm{tp}(m_t, x_t)$. 4 guarantees a $y_t \in M$ such that $\mathrm{tp}(m_t, x_t, y_t) = \tau_t$. Using the effective presentation assumption, F computes such a y_t and outputs it, setting $m_{t+1} = y_t$ for the next step. This creates a concrete trace over $\mathcal{M}$ where each tuple (m_t, x_t, y_t) realizes the type τ_t chosen by G. Since G ensures that the sequence Q_τ satisfies φ^*, theorem 1 implies that the concrete trace satisfies the original data formulas, so F realizes φ. □

$\mathcal{M}$ is fixed, hence the blowup is a function of the number of types and is therefore a constant, but this is only after the reduction into the 3-variable form, so $\mathcal{M}$ is a product structure whose dimension determined by s, ℓ. We obtain:

Corollary 11. LTL $(\mathcal{M})$ *realizability is 2-EXPTIME complete for fixed* $\mathcal{M}, s, \ell$.

Synthesis is now straightforward by construction: given a program that realizes the propositional LTL formula, we reproduce the input encoding and the output decoding according to the types mapped to the propositional variables.

5 Optimizations

5.1 Binary Encoding

One trivial optimization is to encode each 2-type using $\lceil \log_2 |T_2| \rceil$ propositional variables (the type "number" encoded in binary) and similarly for 3-types. Instead of $|T_2| + |T_3|$ propositional variables we obtain only $\lceil \log_2 |T_2| \rceil + \lceil \log_2 |T_3| \rceil$ many, and further we do not need to encode that at exactly one type holds at each point in time as this is automatic from this encoding for both existence and uniqueness.

5.2 Avoiding T_3

Avoiding enumerating T_3 in the worst case is not possible: we have formulas with 3 free variables and we have to keep track of which hold or do not hold at each step. In the worst case, the Boolean algebra generated by those formulas is precisely the one generated by T_3. However, if the generated algebra is coarser, we can take advantage of that.

In what follows let Δ be the set of data subformulas in φ. For $A \in \{\bot, \top\}^{|\Delta|}$, define the Δ-minterm Δ^A by

$$\Delta^A \coloneqq \bigwedge_{\delta | a_\delta = \top} \delta_i\,(m, x, y) \wedge \bigwedge_{\delta | a_\delta = \bot} \neg \delta_i\,(m, x, y)$$

Observe the structure of $\hat{\varphi}$ from section §4: the output propositions Q_τ appear in three places. In φ^*, each data subformula δ was replaced by $\bigvee_{\tau \in \iota(\delta)} Q_\tau$, which simply encodes whether δ holds. In Ψ_I, the output type determines the successor memory type $\mathsf{sh}\,(\tau)\,|_m \in T_1$, which constrains which input types are admissible at the next step. In Ψ_O and Φ_O, the output type must be unique and extend the input type.

So $\hat{\varphi}$ does not need the full type τ. It needs only two things: which of the $|\Delta|$ data subformulas hold, and what the successor memory type is. Replace the $|T_3|$ output propositions Q_τ with $|\Delta| + |T_1|$ propositions: D_i for each $\delta_i \in \Delta$, and R_ρ for each $\rho \in T_1$ encoding the successor's memory type. In φ^*, replace $\bigvee_{\tau \in \iota(\delta_i)} Q_\tau$ with simply D_i. In Ψ_I, replace $\mathsf{sh}\,(\tau)\,|_m$ with ρ. Replace Φ_O with uniqueness for R_ρ:

$$\Phi_O^R \coloneqq \bigvee_{\rho \in T_1} R_\rho \wedge \bigwedge_{\rho_1 \neq \rho_2} \neg \left(R_{\rho_1} \wedge R_{\rho_2}\right)$$

or skip it if using a binary encoding.

So far, not every combination $\left(\rho, \sigma, D_1, \ldots, D_{|\Delta|}\right)$ is witnessed by an actual type. The output could declare $D_1 \wedge D_2$ even when $\delta_1 \wedge \delta_2$ is unsatisfiable. We therefore add the constraint:

$$\Phi_\delta \coloneqq \bigwedge_{\sigma, \rho, A \in \{\bot, \top\}^{|\Delta|}} \left(P_\sigma \wedge R_\rho \wedge \mathcal{D}^A\right) \rightarrow \underbrace{\exists mxy.\sigma\,(m, x) \wedge \rho\,(y) \wedge \Delta^A}_{\text{precomputed}}$$

where $\mathcal{D}^A$ is minterm in the D's in the same fashion as Δ^A. Each infeasible combination contributes one forbidden-pattern clause. The guard Φ_δ subsumes Ψ_O: any feasible $(\sigma, \rho, a_1, \ldots, a_K)$ is witnessed by a τ with $\tau|_{m,x} = \sigma$, so the requirement that the output type extends the input type is already enforced by feasibility. The translated specification now becomes

$$\mathsf{G}\left(\Psi_I \wedge \Phi_I\right) \rightarrow \left(\varphi^* \wedge \mathsf{G}\left(\Phi_O^R \wedge \Phi_\delta\right)\right).$$

The total number of propositional variables is now $|T_1| + |T_2| + |\Delta|$. Combined with the binary encoding optimization, we can reduce it into $\lceil \log_2 |T_1| \rceil + \lceil \log_2 |T_2| \rceil + |\Delta|$.

6 Related Work

LTL modulo theories is an active area of research. We mention only some of the works closest to our approach:

1. RealMT: [14] reduces LTL synthesis modulo $\exists^*\forall^*$-decidable theories to propositional synthesis via Boolean abstraction. However, RealMT variables are *fresh* at each step: the formula $\mathsf{G}\left(y_0 > y_{-1}\right)$ is inexpressible because it compares outputs across time steps. For lookback $\ell = 0$, RealMT subsumes our framework. For $\ell > 0$ our framework seems to be the only result so far providing decidable synthesis with cross-step data references over infinite domains.

2. LTL modulo theories: [12] studies LTL_f modulo theories over *finite* traces, giving only semi-decidability results for satisfiability when data constraints come from decidable SMT theories. [1] extends this to full LTL synthesis over infinite-state arenas using a CEGAR loop: abstract via Boolean predicates, synthesize, check spuriousness, and refine. [5] further presents a safety fragment admitting EXPTIME synthesis. In our case the full safety fragment can easily be supported by writing a fixpoint formula (the standard CPre formulation) over data structure, as mentioned also in section §7.

3. Register automata and data words: synthesis over infinite alphabets has been studied through register automata. [9] considers synthesis of register transducers over the dense linear order, using a finite number of registers to store data values. [15] studies synthesis for register automata specifications with universality constraints. They prove both decidability and undecidability results, depending on register bounds. In our framework, r registers over $(\mathbb{Q}, <)$ (which is an ω-categorical structure) correspond to lookback $\ell = r$ with $s = 1$ stream. The type spaces grow with ℓ but remain finite for each fixed ℓ. The register automata approach does not directly generalize to richer structures while our framework handles any ω-categorical $\mathcal{M}$.

4. Data-word logics: Freeze LTL [7] extends LTL with a freeze quantifier that stores the current data value in a register. Over infinite data domains, the satisfiability problem for freeze LTL is already undecidable with two registers. Our bounded lookback ℓ avoids this: the number of simultaneously accessible past values is fixed, preventing the encoding of counter machines. This is a deliberate design choice that enables decidability.

5. Orbit-Finiteness: [4] develops an automata theory parameterized by a symmetry group G acting on an infinite alphabet. An automaton is *orbit-finite* if its state set has finitely many G-orbits. When $G = \mathrm{Aut}\left(\mathcal{M}\right)$ for an ω-categorical $\mathcal{M}$, their orbit-finite sets are exactly our

type spaces $T_k = M^k/\mathrm{Aut}(\mathcal{M})$ - not an analogy but an identity via the Ryll-Nardzewski theorem, even though [4] does not mention ω-categoricity. They prove decidability of emptiness for orbit-finite automata but undecidability of universality for the nondeterministic case. The main difference between their and our approaches is that they study language-theoretic properties (emptiness, inclusion, determinization) while we study synthesis. Our equirealizability theorem is specific to the synthesis setting and seems to have no direct counterpart in the orbit-finite automata literature.

7 Conclusion

We have shown that ω-categoricity gives decidable LTL+P realizability and synthesis, while allowing data variables to interact over bounded lookback, and while showing that from a theoretical point of view, when compared to propositional LTL, only the lookback increases the complexity. The types are the only $\mathcal{M}$-dependent data. Once computed, the main realizability and synthesis algorithms are purely propositional, with type enumeration being main blowup.

The main beneficiaries of ω-categoricity are fragments of LTL for which realizability and synthesis can be translated directly into fixpoint operators, in which case no type enumeration nor Δ-minterms are needed. A direct corollary of theorem 1 is that in ω-categorical structures, fixpoint operators can be eliminated. Therefore in such fragments of LTL, there is no need to use propositional encoding, and the problem can be decided directly using fixed points.

Acknowledgments

I would like to thank Enrico Franconi, Lucca Tiemens, and Paweł Parys, for plenty of useful discussions.

Intellectual Property

Some of the methods discussed here are patented.

References

[1] Shaun Azzopardi, Alessandro Cimatti, Luca Geatti, and Nicola Gigante. Full LTL synthesis over infinite-state arenas. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2024)*, volume 14572 of *Lecture Notes in Computer Science*, pages 276–295. Springer, 2024.

[2] Ashwin Bhaskar and M. Praveen. Realizability problem for constraint LTL. In *29th International Symposium on Temporal Representation and Reasoning (TIME 2022)*, volume 247 of *LIPIcs*, pages 8:1–8:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

[3] Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Transactions on Computational Logic*, 12(4):27:1–27:26, 2011. Conference version: LICS 2006.

[4] Mikołaj Bojańczyk, Bartek Klin, and Sławomir Lasota. Automata theory in nominal sets. *Logical Methods in Computer Science*, 10(3), 2014.

[5] Alessandro Cimatti, Luca Geatti, Nicola Gigante, Angelo Montanari, and Stefano Tonetta. Extended bounded response LTL: A new safety fragment for efficient reactive synthesis. *Formal Methods in System Design*, 64(1):1–49, 2024.

[6] Stéphane Demri and Deepak D'Souza. An automata-theoretic approach to constraint LTL. *Information and Computation*, 205(3):380–415, 2007.

[7] Stéphane Demri and Ranko Lazić. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10(3):16:1–16:30, 2009.

[8] Léo Exibard, Emmanuel Filiot, and Ayrat Khalimov. A generic solution to register-bounded synthesis with an application to discrete orders. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *LIPIcs*, pages 122:1–122:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

[9] Léo Exibard, Emmanuel Filiot, Ayrat Khalimov, and Pierre-Alain Reynier. Church synthesis on register automata over linearly ordered data domains. In *38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*, volume 187 of *LIPIcs*, pages 28:1–28:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

[10] Bernd Finkbeiner, Philippe Heim, and Noemi Passing. Temporal stream logic modulo theories. In *Foundations of Software Science and Computation Structures (FoSSaCS 2022)*, volume 13242 of *Lecture Notes in Computer Science*, pages 325–346. Springer, 2022.

[11] Bernd Finkbeiner, Felix Klein, Ruzica Piskac, and Mark Santolucito. Temporal stream logic: Synthesis beyond the bools. In *Computer Aided Verification - 31st International Conference, CAV 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 609–629. Springer, 2019.

[12] Luca Geatti, Alessandro Gianola, and Nicola Gigante. Linear temporal logic modulo theories over finite traces. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2641–2647. ijcai.org, 2022.

[13] Wilfrid Hodges. *Model Theory*, volume 42 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1993.

[14] Andreas Katis, Anastasia Mavridou, Thomas Reinbacher, and Cesare Tinelli. Realizability modulo theories. *International Journal on Software Tools for Technology Transfer*, 27:169–189, 2025.

[15] Ayrat Khalimov and Bernd Finkbeiner. Register-bounded synthesis. In *Computer Aided Verification - 32nd International Conference, CAV 2020, Proceedings, Part II*, volume 12225 of *Lecture Notes in Computer Science*, pages 25–45. Springer, 2020.

[16] Ayrat Khalimov, Benedikt Maderbacher, and Roderick Bloem. Bounded synthesis of register transducers. In *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*, pages 494–510. Springer, 2018.