

Tracy, Traces, and Transducers: Computable Counterexamples and Explanations for HyperLTL Model-Checking*

Sarah Winter (IRIF & Université Paris Cité, Paris, France)
Martin Zimmermann (Aalborg University, Aalborg, Denmark)

Abstract

HyperLTL model-checking enables the automated verification of information-flow properties for security-critical systems. However, it only provides a binary answer. Here, we introduce two paradigms to compute counterexamples and explanations for HyperLTL model-checking, thereby considerably increasing its usefulness. Both paradigms are based on the maxim “counterexamples/explanations are Skolem functions for the existentially quantified trace variables”.

Our first paradigm is complete (everything can be explained), but restricted to ultimately periodic system traces. The second paradigm works with (Turing machine) computable Skolem functions and is therefore much more general, but also shown incomplete (not everything can computably be explained). Finally, we prove that it is decidable whether a given finite transition system and a formula have computable Skolem functions witnessing that the system satisfies the formula. Our algorithm also computes transducers implementing computable Skolem functions, if they exist.

1 Introduction

Prologue. Tracy sits in her office and needs to print her latest travel reimbursement claim. After hitting the print button, she walks to the printer room only to find out that the document has not been printed. So, she walks back to her office, hits the print button again, walks to the printer and is slightly surprised to find her document. Sometimes Tracy wonders whether the print system is nondeterministic. If only there was a way to find out.

Information-flow properties, which are crucial in the specification of security-critical systems, require the simultaneous reasoning about multiple executions of a system. However, most classical specification languages like LTL and CTL* refer to a single execution trace at a time. Clarkson and Schneider [10] coined the term *hyperproperties* for properties that require the reasoning about multiple traces. Just like ordinary trace and branching-time properties, hyperproperties can be specified using temporal logics, e.g., HyperLTL and HyperCTL* [9], expressive, but intuitive specification languages that are able to express typical information-flow properties such as noninterference, noninference, declassification, and input determinism. Due to their practical relevance and theoretical elegance, hyperproperties and their specification languages have received considerable attention during the last decade.

HyperLTL is obtained by extending LTL [28], the most influential specification language for linear-time properties, by trace quantifiers to refer to multiple executions of a system. Hence, a HyperLTL formula is indeed evaluated over a set of traces, which forms the universe for the quantifiers. For example, the HyperLTL formula $\varphi_{\text{id}} = \forall \pi, \pi'. \mathbf{G}(i_\pi \leftrightarrow i_{\pi'}) \rightarrow \mathbf{G}(o_\pi \leftrightarrow o_{\pi'})$ expresses input determinism, i.e., every pair of traces that always has the same input (represented by the proposition i) also always has the same

*Supported by DIREC - Digital Research Centre Denmark.

output (represented by the proposition o). Having learned about HyperLTL, Tracy wonders whether she can formally prove that the print system violates φ_{id} .

In this work, we focus on the model-checking problem for HyperLTL, which intuitively asks whether a given (finite model of a) system satisfies a given HyperLTL specification. This problem is decidable, albeit TOWER-complete [30, 27].

But the model-checking problem as described above is “just” a decision problem, i.e., the user only learns whether the system satisfies the specification or not, but not the *reason* it does or does not. It has been argued that this binary answer is in general not useful [26]: Most real-life systems are too complex to be modelled faithfully by a finite transition system. Hence, one always checks an abstraction, not the actual system. Then, a positive answer to the model-checking problem does not show that the actual system is correct, bugs in it might have been abstracted away when constructing a finite transition system modelling it. The actual killer application of model-checking is the automated generation of counterexamples in case the specification is not satisfied by the abstraction. Given a counterexample in the abstraction one can then check whether this (erroneous) behaviour also exists in the actual system, or whether it was introduced during the abstraction. In the latter case, the abstraction has to be refined and checked again. But if the erroneous behaviour can be found in the actual system, then this bug can be fixed in the actual system.

But what is a counterexample in HyperLTL model-checking? For the formula φ_{id} expressing input determinism this is straightforward: if a transition system does not satisfy the formula, then it has two traces that coincide on their input, but not on their output. However, the situation becomes more interesting in the presence of existentially quantified variables and quantifier alternations. Consider, for example, a formula of the form $\varphi = \exists\pi\forall\pi'. \psi$ with quantifier-free ψ and let \mathfrak{T} be a transition system with set $\text{Tr}(\mathfrak{T})$ of traces. If $\mathfrak{T} \not\models \varphi$, then for every choice of $t \in \text{Tr}(\mathfrak{T})$ there is a $t' \in \text{Tr}(\mathfrak{T})$ such that the variable assignment $\{\pi \mapsto t, \pi' \mapsto t'\}$ does not satisfy ψ . Thus, a counterexample is described by a Skolem function $f: \text{Tr}(\mathfrak{T}) \rightarrow \text{Tr}(\mathfrak{T})$ for the existentially quantified variable π' in the negation $\forall\pi\exists\pi'. \neg\psi$ of φ . It gives, for every choice t for the existentially quantified π in φ a trace $f(t)$ for the universally quantified π' in φ such that $\{\pi \mapsto t, \pi' \mapsto f(t)\} \models \neg\psi$, i.e., $\{\pi \mapsto t, \pi' \mapsto f(t)\} \not\models \psi$, thereby explaining for every choice of t why it is not a good one. The maxim “counterexamples are Skolem functions for existentially quantified variables in the negation of the specification” is true for arbitrary formulas. But before we explore this approach further, let us first consider a second application.

Explainability, the need to explain to, e.g., users, customers, and regulators, what a system does, is an aspect of system design that gains more and more significance. This is in particular true when it comes to systems designed by algorithms, e.g., machine-learning or synthesis. For any nontrivial such system, it is impossible for humans to develop an explanation of their behaviour or a witness for their correctness. This is a major obstacle preventing the wide-spread use of (unexplained) machine-generated software in safety-critical applications [3]. Also here, HyperLTL model-checking can be useful: Assuming the system is supposed to satisfy a HyperLTL specification and indeed does so, then Skolem functions “explain” why the specification is satisfied.

In this work, we are interested in computing counterexamples/explanations for HyperLTL. Before we do so, let us remark that these counterexamples are really explanations for the negation of the specification, as we have seen above. Hence, in the following we will focus on explanations, as this setting spares us from dealing with a negation. Also, let us remark that for every transition system \mathfrak{T} and every HyperLTL formula φ , we either have $\mathfrak{T} \models \varphi$ or $\mathfrak{T} \models \neg\varphi$. Hence, our framework will either explain why \mathfrak{T} satisfies φ or explain why \mathfrak{T} satisfies $\neg\varphi$, i.e., explain why \mathfrak{T} does not satisfy φ .

In general, we are given a transition system \mathfrak{T} and a HyperLTL formula φ such that $\mathfrak{T} \models \varphi$, and we want to compute Skolem functions for the existentially quantified variables in φ . Note that the actual explanation-phase employing the Skolem functions is an interactive process between the user (i.e., Tracy) and these functions: Tracy has to specify choices for the universally quantified variables, which are then feed into the Skolem functions, yielding choices for the existentially quantified variables such that the combination of all of these traces satisfies the quantifier-free part of the specification.

However, as the inputs to the Skolem functions are (infinite) traces of \mathfrak{T} , their domain may be uncountable. Hence, we need to discuss how to represent such functions, as, due to a simple counting argument, not each

possible Skolem function is finitely representable. Here, we propose and investigate two paradigms.

The up-paradigm. In the up-paradigm, we restrict ourselves to ultimately periodic traces, which are finitely representable, and do not actually compute the full Skolem functions, but only restrictions of the Skolem functions that suffice to handle ultimately periodic inputs. Such traces are particularly well-suited for human inspection, which is not always true for non-ultimately periodic traces.

The cs-paradigm. On the other hand, the cs-paradigm is much more general: Here, we work with Skolem functions that are computable by Turing machines (in a very natural sense). Continuing previous work by Filiot and Winter [13], we show that such functions are actually implementable by a much simpler machine model, i.e., word-to-word transducers with bounded delay between input and output. This model allows the effective computation and simulation of computable Skolem functions.

Beyond expressiveness, there is another key difference between the two paradigms: Computable Skolem functions are always continuous: Intuitively, if two inputs coincide on a “long” prefix, then the corresponding outputs also coincide on a “long” prefix. However, explanations in the up-paradigm are not continuous. For example, consider the formula $\forall\pi\exists\pi'. \text{true}$. Then, a Skolem function for π' could on input $\emptyset^n\{a\}^\omega$ return \emptyset^ω if n is even and $\{a\}^\omega$ if n is odd. Hence, the sequence $(\emptyset^n\{a\}^\omega)_{n\in\mathbb{N}}$ of inputs converges, but the sequence of outputs does not. But, continuity makes explanations easier to understand: settled outputs are never revoked. However, it is straightforward to construct a pair (\mathfrak{T}, φ) with $\mathfrak{T} \models \varphi$ that does not have continuous explanations (see Theorem 2). Hence, the cs-paradigm is incomplete, unlike the up-paradigm, in which every such pair has an explanation.

This incompleteness means that deciding whether a computable explanation exists is non-trivial (in the sense that the answer is not always “yes”, as in the up-paradigm). Combining techniques developed in the theory of uniformization [13], delay games [24], and concurrent multiplayer games with hierarchical imperfect information [4], we show that this problem is decidable. As a byproduct, we present an algorithm that generates a computable explanation (implemented by transducers), whenever one exists.

The cs-paradigm can be used in an on-the-fly manner in which Tracy specifies, letter by letter, traces for the universally quantified variables of the specification and the transducers produce (possibly with some delay) corresponding traces for the existentially quantified variables. This process allows Tracy to understand how the evolution of the traces for the universally quantified variables influences the evolution of the existentially quantified variables.

Alternatively, the cs-paradigm can also be used by feeding the transducers with infinite inputs (traces for the universally quantified variables), which yields corresponding outputs (traces for the existentially quantified variables). As the transducers have bounded delay, Tracy only has to provide prefixes of length $n+d$ as inputs to obtain prefixes of length n for the existentially quantified variables, where d is the delay of the transducers.

2 Preliminaries¹

We denote the set of nonnegative integers by \mathbb{N} . The domain of a partial function $f: A \rightarrow B$ is denoted by $\text{dom}(f) = \{a \in A \mid f(a) \text{ is defined}\}$. More generally, we denote the domain $\{a \in A \mid (a, b) \in R \text{ for some } b \in B\}$ of a relation $R \subseteq A \times B$ by $\text{dom}(R)$.

Languages and Transition Systems. An alphabet is a nonempty finite set. The sets of finite and infinite words over an alphabet Σ are denoted by Σ^* and Σ^ω , respectively. The length of a finite word w is denoted by $|w|$. An infinite word $w \in \Sigma^\omega$ is ultimately periodic if there are finite words $x, y \in \Sigma^*$ such that

¹We like the twentieth letter of the alphabet. In fact, we like it so much that we use t to denote traces, T to denote sets of traces, \mathfrak{T} to denote transition systems, and \mathcal{T} to denote transducers. We hope this footnote will help the reader keeping track of them.

$w = xy^\omega$. Given n infinite words w_0, \dots, w_{n-1} , let their merge (also known as zip) be defined as

$$\text{mrg}(w_0, \dots, w_{n-1}) = \begin{pmatrix} w_0(0) \\ \vdots \\ w_{n-1}(0) \end{pmatrix} \begin{pmatrix} w_0(1) \\ \vdots \\ w_{n-1}(1) \end{pmatrix} \begin{pmatrix} w_0(2) \\ \vdots \\ w_{n-1}(2) \end{pmatrix} \cdots \in (\Sigma^n)^\omega.$$

We define $\text{mrg}(w_0, \dots, w_{n-1})$ for finite words w_0, \dots, w_n of the same length analogously.

The set of prefixes of an infinite word $w = w(0)w(1)w(2) \cdots \in \Sigma^\omega$ is $\text{Prfs}(w) = \{w(0) \cdots w(i-1) \mid i \geq 0\}$, which is lifted to languages $L \subseteq \Sigma^\omega$ via $\text{Prfs}(L) = \bigcup_{w \in L} \text{Prfs}(w)$. A language $L \subseteq \Sigma^\omega$ is closed if $\{w \in \Sigma^\omega \mid \text{Prfs}(w) \subseteq \text{Prfs}(L)\} \subseteq L$.

Throughout this paper, we fix a finite set AP of atomic propositions. A transition system $\mathfrak{T} = (V, E, v_I, \lambda)$ consists of a finite set V of vertices, a set $E \subseteq V \times V$ of (directed) edges, an initial vertex $v_I \in V$, and a labelling $\lambda: V \rightarrow 2^{\text{AP}}$ of the vertices by sets of atomic propositions. We assume that every vertex has at least one outgoing edge. A path ρ through \mathfrak{T} is an infinite sequence $\rho = v_0 v_1 v_2 \cdots$ of vertices with $v_0 = v_I$ and $(v_n, v_{n+1}) \in E$ for every $n \geq 0$. The trace of ρ is defined as $\lambda(\rho) = \lambda(v_0)\lambda(v_1)\lambda(v_2) \cdots \in (2^{\text{AP}})^\omega$. The set of traces of \mathfrak{T} is $\text{Tr}(\mathfrak{T}) = \{\lambda(\rho) \mid \rho \text{ is a path of } \mathfrak{T}\}$.

Remark 1. *The following facts follow directly from the definition of closed languages.*

1. *Let \mathfrak{T} be a transition system. Then, $\text{Tr}(\mathfrak{T})$ is closed.*
2. *If $L_0, \dots, L_{n-1} \subseteq \Sigma^\omega$ are closed, then so is $\{\text{mrg}(w_0, \dots, w_{n-1}) \mid w_i \in L_i \text{ for } 0 \leq i < n\}$.*

HyperLTL. The formulas of HyperLTL are given by the grammar

$$\varphi ::= \exists \pi. \varphi \mid \forall \pi. \varphi \mid \psi \quad \psi ::= a_\pi \mid \neg \psi \mid \psi \vee \psi \mid \mathbf{X} \psi \mid \psi \mathbf{U} \psi$$

where a ranges over AP and where π ranges over a fixed countable set \mathcal{V} of (trace) variables. Conjunction (\wedge), exclusive disjunction (\oplus), implication (\rightarrow), and equivalence (\leftrightarrow) are defined as usual, and the temporal operators “eventually” (**F**) and “always” (**G**) are derived as $\mathbf{F} \psi = \neg \psi \mathbf{U} \psi$ and $\mathbf{G} \psi = \neg \mathbf{F} \neg \psi$. A sentence is a formula without free variables, which are defined as expected.

The semantics of HyperLTL is defined with respect to a trace assignment, a partial mapping $\Pi: \mathcal{V} \rightarrow (2^{\text{AP}})^\omega$. The assignment with empty domain is denoted by Π_\emptyset . Given a trace assignment Π , a variable π , and a trace t we denote by $\Pi[\pi \rightarrow t]$ the assignment that coincides with Π everywhere but at π , which is mapped to t . Furthermore, $\Pi[j, \infty)$ denotes the trace assignment mapping every π in Π 's domain to $\Pi(\pi)(j)\Pi(\pi)(j+1)\Pi(\pi)(j+2) \cdots$, the suffix of $\Pi(\pi)$ starting at position j .

For sets T of traces and trace assignments Π we define

- $(T, \Pi) \models a_\pi$ if $a \in \Pi(\pi)(0)$,
- $(T, \Pi) \models \neg \psi$ if $(T, \Pi) \not\models \psi$,
- $(T, \Pi) \models \psi_1 \vee \psi_2$ if $(T, \Pi) \models \psi_1$ or $(T, \Pi) \models \psi_2$,
- $(T, \Pi) \models \mathbf{X} \psi$ if $(T, \Pi[1, \infty)) \models \psi$,
- $(T, \Pi) \models \psi_1 \mathbf{U} \psi_2$ if there is a $j \geq 0$ such that $(T, \Pi[j, \infty)) \models \psi_2$ and for all $0 \leq j' < j$: $(T, \Pi[j', \infty)) \models \psi_1$,
- $(T, \Pi) \models \exists \pi. \varphi$ if there exists a trace $t \in T$ such that $(T, \Pi[\pi \rightarrow t]) \models \varphi$, and
- $(T, \Pi) \models \forall \pi. \varphi$ if for all traces $t \in T$: $(T, \Pi[\pi \rightarrow t]) \models \varphi$.

We say that T satisfies a sentence φ if $(T, \Pi_\emptyset) \models \varphi$. In this case, we write $T \models \varphi$ and say that T is a model of φ . A transition system \mathfrak{T} satisfies φ , written $\mathfrak{T} \models \varphi$, if $\text{Tr}(\mathfrak{T}) \models \varphi$. Although HyperLTL sentences are required to be in prenex normal form, they are closed under Boolean combinations, which can be easily seen by transforming such a formula into an equivalent formula in prenex normal form. In particular, the negation $\neg \varphi$ of a sentence φ satisfies $T \models \neg \varphi$ if and only if $T \not\models \varphi$. Also, note that the statement $(T, \Pi) \models \psi$ for quantifier-free formulas ψ is independent of T . Hence, we often just write $\Pi \models \psi$ for the sake of readability.



Figure 1: The transition system for Example 2.

Skolem Functions for HyperLTL. Let $\varphi = Q_0\pi_0 \cdots Q_{k-1}\pi_{k-1} \cdot \psi$ be a HyperLTL sentence such that ψ is quantifier-free and let T be a set of traces. Moreover, let $i \in \{0, 1, \dots, k-1\}$ be such that $Q_i = \exists$ and let $U_i = \{j < i \mid Q_j = \forall\}$ be the indices of the universal quantifiers preceding Q_i . Furthermore, let $f_i: T^{|U_i|} \rightarrow T$ for each such i (note that f_i is a constant, if U_i is empty). We say that a variable assignment Π with $\text{dom}(\Pi) \supseteq \{\pi_0, \pi_1, \dots, \pi_{k-1}\}$ is consistent with the f_i if $\Pi(\pi_i) \in T$ for all i with $Q_i = \forall$ and $\Pi(\pi_i) = f_i(\Pi(\pi_{i_0}), \Pi(\pi_{i_1}), \dots, \Pi(\pi_{i_{|U_i|-1}}))$ for all i with $Q_i = \exists$, where $U_i = \{i_0 < i_1 < \dots < i_{|U_i|-1}\}$. If $\Pi \models \psi$ for each Π that is consistent with the f_i , then we say that the f_i are Skolem functions witnessing $T \models \varphi$.

Remark 2. $T \models \varphi$ if and only if there are Skolem functions for the existentially quantified variables of φ that witness $T \models \varphi$.

Note that only traces for universal variables are inputs for Skolem functions, but not those for existentially quantified variables. As usual, this is not a restriction, as the inputs of a Skolem function for an existentially quantified variable π_i is a superset of the inputs of a Skolem function for another existentially quantified variable π_j with $j < i$.

Example 1. Let $\varphi = \forall\pi\exists\pi_1\exists\pi_2. \mathbf{G}(a_\pi \leftrightarrow (a_{\pi_1} \oplus a_{\pi_2}))$. We have $(2^{\{a\}})^\omega \models \varphi$. Now, for every function $f_1: (2^{\{a\}})^\omega \rightarrow (2^{\{a\}})^\omega$, there is a function $f_2: (2^{\{a\}})^\omega \rightarrow (2^{\{a\}})^\omega$ such that f_1, f_2 are Skolem functions witnessing $(2^{\{a\}})^\omega \models \varphi$, i.e., we need to define f_2 such that $(f_2(t))(n) = (f_1(t))(n)$ for all $n \in \mathbb{N}$ such that $t(n) = \emptyset$ and $(f_2(t))(n) = \overline{(f_1(t))(n)}$ for all $n \in \mathbb{N}$ such that $t(n) = \{a\}$, where $\overline{\{a\}} = \emptyset$ and $\overline{\emptyset} = \{a\}$. Hence, f_2 depends on f_1 , but the value of $f_1(t)$ (for the existentially quantified π_1) does not need to be an input to f_2 , it can be determined from the input t for the universally quantified π . This is not surprising, but needs to be taken into account in our constructions.

We conclude by informally illustrating the two paradigms described in the introduction.

Example 2. Consider the transition system \mathfrak{T} given in Figure 1 and the sentence $\varphi = \forall\pi_0\exists\pi_1\forall\pi_2\exists\pi_3. \psi$ with

$$\psi = [(\mathbf{G} \mathbf{F} a_{\pi_0}) \leftrightarrow (\mathbf{G} \mathbf{F} a_{\pi_1})] \wedge [(\mathbf{G} \mathbf{F} a_{\pi_0} \wedge \mathbf{G} \mathbf{F} a_{\pi_2}) \leftrightarrow (\mathbf{G} \mathbf{F} a_{\pi_3})] \wedge [\mathbf{G} \neg(a_{\pi_1} \wedge a_{\pi_3})]$$

and note that we have $\mathfrak{T} \models \varphi$.

In the up-paradigm, Tracy could, for example, pick the ultimately periodic trace $t_0 = (\emptyset\{a\})^\omega$ for π_0 and a Skolem function yields the trace $t_1 = (\emptyset\{a\}\emptyset\emptyset)^\omega$ for π_1 . Then, Tracy could pick $t_2 = t_0$ and a Skolem function yields the trace $t_3 = (\emptyset\emptyset\emptyset\{a\})^\omega$. The assignment $\{\pi_0 \mapsto t_0, \dots, \pi_3 \mapsto t_3\}$ satisfies ψ . As in Example 1, the choice of t_3 depends not only on the choices for the universally quantified variables π_0 and π_2 , but also on the choice for π_1 , as the last conjunct of ψ requires that a is never simultaneously true in π_1 and π_3 .

In the cs-paradigm, we could obtain a transducer for π_1 mapping $t_0(0)t_0(1)t_0(2) \cdots \in \text{Tr}(\mathfrak{T})$ to the trace $t_1(0)t_1(1)t_1(2) \cdots \in \text{Tr}(\mathfrak{T})$ where

$$t_1(n) = \begin{cases} \{a\} & \text{if } n \geq 4, n \bmod 4 = 1, \text{ and } t_0(n-3) \cdots t_0(n) \text{ contains an } \{a\}, \\ \emptyset & \text{otherwise,} \end{cases}$$

and a transducer for π_3 that maps $(t_0(0)t_0(1)t_0(2) \cdots, t_2(0)t_2(1)t_2(2) \cdots) \in \text{Tr}(\mathfrak{T}) \times \text{Tr}(\mathfrak{T})$ to the trace $t_3(0)t_3(1)t_3(2) \cdots \in \text{Tr}(\mathfrak{T})$ where

$$t_3(n) = \begin{cases} \{a\} & \text{if } n \geq 3, n \bmod 4 = 3, \text{ and } t_0(n'+1) \cdots t_0(n) \text{ and } t_2(n'+1) \cdots t_2(n) \\ & \text{both contain an } \{a\}, \text{ where } n' < n \text{ is maximal with } t_3(n') = \{a\}, \\ \emptyset & \text{otherwise.} \end{cases}$$

Both functions described above are indeed implementable by a transducer (see Section 8 of the appendix for formal definitions) and witness $\text{Tr}(\mathfrak{T}) \models \varphi$.

Parity Automata. We end this section by introducing ω -automata, which we use to represent languages of (tuples of) traces. We use parity automata here, as they are (unlike Büchi automata) determinizable and recognize all ω -regular languages [18]. This is useful in proofs.

A (deterministic) parity automaton $\mathcal{P} = (Q, \Sigma, q_I, \delta, \Omega)$ consists of a finite set Q of states containing the initial state $q_I \in Q$, an alphabet Σ , a transition function $\delta: Q \times \Sigma \rightarrow Q$, and a coloring $\Omega: Q \rightarrow \mathbb{N}$ of its states by natural numbers. Let $w = w(0)w(1)w(2) \cdots \in \Sigma^\omega$. The run of \mathcal{P} on w is the sequence $q_0q_1q_2 \cdots$ with $q_0 = q_I$ and $q_{n+1} = \delta(q_n, w(n))$ for all $n \geq 0$. A run $q_0q_1q_2 \cdots$ is (parity) accepting if the maximal color appearing infinitely often in the sequence $\Omega(q_0)\Omega(q_1)\Omega(q_2) \cdots$ is even. The language (parity) recognized by \mathcal{P} , denoted by $L(\mathcal{P})$, is the set of infinite words over Σ such that the run of \mathcal{P} on w is accepting.

Proposition 1 (See, e.g., [18]).

1. Parity automata are closed under all Boolean operations.
2. The nonemptiness problem for parity automata “Given a parity automaton \mathcal{P} , is $L(\mathcal{P})$ nonempty” is in PTIME. Furthermore, if $L(\mathcal{P})$ is nonempty, one can compute in polynomial time an ultimately periodic word in $L(\mathcal{P})$.

3 Automata-based HyperLTL Model-Checking

In this section, we recall the automata-based approach to HyperLTL model-checking [9, 15]. Throughout this subsection, we fix a HyperLTL sentence $\varphi = Q_0\pi_0Q_1\pi_1 \cdots Q_{k-1}\pi_{k-1}.\psi$ and define $\varphi_i = Q_i\pi_iQ_{i+1}\pi_{i+1} \cdots Q_{k-1}\pi_{k-1}.\psi$ for $i \in \{0, 1, \dots, k-1\}$ and $\varphi_k = \psi$. Note that $\varphi_0 = \varphi$ and that the free variables of each φ_i with $i \in \{0, 1, \dots, k\}$ are exactly π_0, \dots, π_{i-1} .

Proposition 2 ([9, 15]). *Let \mathfrak{T} be a transition system. For every $i \in \{0, 1, \dots, k\}$ there is an (effectively constructible) parity automaton \mathcal{P}_i such that*

$$L(\mathcal{P}_i) = \{\text{mrg}(\Pi(\pi_0), \dots, \Pi(\pi_{i-1})) \mid \Pi(\pi_j) \in (2^{\text{AP}})^\omega \text{ for } 0 \leq j < i \text{ and } (\text{Tr}(\mathfrak{T}), \Pi) \models \varphi_i\}.$$

Note that \mathcal{P}_0 is a parity automaton over a singleton alphabet (containing the empty tuple) that has an accepting run if and only if $\mathfrak{T} \models \varphi$. Hence, the HyperLTL model-checking problem can be solved by constructing \mathcal{P}_0 and checking it for nonemptiness. However, \mathcal{P}_0 may be large: each quantifier alternation requires a complementation and projection (which introduces nondeterminism), implying that each complementation may be exponential. This blowup is unavoidable, as HyperLTL model-checking is TOWER-complete [30, 27].

Remark 3. *Note that we may have $\text{mrg}(\Pi(\pi_0), \dots, \Pi(\pi_{i-1})) \in L(\mathcal{P}_i)$, but $\Pi(\pi_j) \notin \text{Tr}(\mathfrak{T})$. But oftentimes we want to restrict ourselves to traces from \mathfrak{T} . Hence, for each $i \in \{0, 1, \dots, k\}$ let $\mathcal{P}_i^{\mathfrak{T}}$ be a parity automaton for the language $\{\text{mrg}(t_0, \dots, t_{i-1}) \in L(\mathcal{P}_i) \mid t_j \in \text{Tr}(\mathfrak{T}) \text{ for all } 0 \leq j < i\}$, which can be effectively computed from \mathcal{P}_i and \mathfrak{T} .*

4 The up-Paradigm

Recall that in the up-paradigm, we only work with ultimately periodic traces, i.e., we essentially work with restrictions of Skolem functions to such traces. This enables an effective computation, as such traces can be finitely represented. Also, the approach is complete in the sense that $\mathfrak{T} \models \varphi$ has such an explanation, for every transition system \mathfrak{T} and every sentence φ with $\mathfrak{T} \models \varphi$. On the other hand, the paradigm is limited to ultimately periodic traces, i.e., any nonperiodic behavior cannot be analyzed nor explained in this paradigm.

Using the parity automata constructed in Section 3, we present here the up-paradigm for computing explanations for HyperLTL. For the remainder of this section, we fix a transition system \mathfrak{T} and a HyperLTL sentence $\varphi = Q_0\pi_0 Q_1\pi_1 \cdots Q_{k-1}\pi_{k-1} \cdot \psi$ such that $\mathfrak{T} \models \varphi$. As before, we define $\varphi_i = Q_i\pi_i Q_{i+1}\pi_{i+1} \cdots Q_{k-1}\pi_{k-1} \cdot \psi$ for $i \in \{0, \dots, k-1\}$ and $\varphi_k = \psi$. For the sake of simplicity, we assume that $k-1$ is odd and that the Q_i with even (odd) i are universal (existential).² Hence, we have $\varphi = \forall\pi_0 \exists\pi_1 \cdots \forall\pi_{k-2} \exists\pi_{k-1} \cdot \psi$. We will use the parity automata $\mathcal{P}_i^{\mathfrak{T}}$ from Remark 3 satisfying

$$L(\mathcal{P}_i^{\mathfrak{T}}) = \{\text{mrg}(\Pi(\pi_0), \dots, \Pi(\pi_{i-1})) \mid \Pi(\pi_j) \in \text{Tr}(\mathfrak{T}) \text{ for all } 0 \leq j < i \text{ and } (\text{Tr}(\mathfrak{T}), \Pi) \models \varphi_i\}.$$

Consider the following procedure:

1. Tracy picks an ultimately periodic $t_0 \in \text{Tr}(\mathfrak{T})$ for the universally quantified variable π_0 .
2. The language $L_1(t_0) = \{t \in \text{Tr}(\mathfrak{T}) \mid \text{mrg}(t_0, t) \in L(\mathcal{P}_2^{\mathfrak{T}})\}$ is recognized by a parity automaton which can be obtained by taking the product of $\mathcal{P}_2^{\mathfrak{T}}$ and a parity automaton for the language $\{\text{mrg}(t_0, t) \mid t \in (2^{\text{AP}})^\omega\}$, which can easily be constructed from t_0 .
Furthermore, as we have $\mathfrak{T} \models \forall\pi_0 \exists\pi_1 \cdot \varphi_2$, $L_1(t_0)$ is nonempty for every choice of $t_0 \in \text{Tr}(\mathfrak{T})$. Formally, it contains, all those $t \in \text{Tr}(\mathfrak{T})$ such that $(\text{Tr}(\mathfrak{T}), \{\pi_0 \mapsto t_0, \pi_1 \mapsto t\}) \models \varphi_2$. Thus, Tracy (or an algorithm solving the nonemptiness problem) can generate an ultimately periodic $t_1 \in L_1(t_0)$ for the existentially quantified variable π_1 .
3. Tracy picks an ultimately periodic $t_2 \in \text{Tr}(\mathfrak{T})$ for the universally quantified variable π_2 .
4. The language $L_3(t_0, t_1, t_2) = \{t \in \text{Tr}(\mathfrak{T}) \mid \text{mrg}(t_0, t_1, t_2, t) \in L(\mathcal{P}_4^{\mathfrak{T}})\}$ is again recognized by some parity automaton and nonempty. Thus, Tracy (or an algorithm solving the nonemptiness problem) can pick an ultimately periodic $t_3 \in L_3(t_0, t_1, t_2)$ for the existentially quantified variable π_3 .
5. We proceed until traces t_0, t_1, \dots, t_{k-1} have been picked.

These traces have the property that the variable assignment mapping each π_i to t_i satisfies ψ . Thus, the procedure above explains $\mathfrak{T} \models \psi$.

Note that whenever we have $\mathfrak{T} \models \varphi$, then the procedure described above is applicable. In particular, there are always ultimately periodic traces to pick from.³ Thus, in this sense, the up-paradigm is complete.

Theorem 1. *Let φ be a HyperLTL sentence and \mathfrak{T} a transition system. If $\mathfrak{T} \models \varphi$, then this fact has an explanation in the up-paradigm.*

Proof. An induction shows that each $L_i(t_0, \dots, t_{i-1})$ for odd i is a nonempty language recognized by a parity automaton. Hence, it contains, due to Proposition 1, an ultimately periodic word. This implies that the procedure never gets stuck and one indeed obtains traces for all variables. \square

Thus, in the up-paradigm, if we have $\mathfrak{T} \models \varphi$, then this fact has an explanation. However, in this paradigm, explanations are very restricted, i.e., we are working with the restrictions of Skolem functions to ultimately periodic traces. On the one hand, this is very natural and user-friendly, as the representation, reasoning, and human inspection of traces that are not ultimately periodic is challenging. However, not every trace is ultimately periodic. Further, as argued in the introduction, the up-paradigm does not give continuous explanations.

²The following reasoning can easily be extended to general sentences with arbitrary quantifier prefixes, albeit at the cost of more complex notation.

³Note that this is true, even though there are satisfiable HyperLTL sentences that do not have any model containing an ultimately periodic trace [17]. However, as every finite transition system contains an ultimately periodic trace, such sentences are not satisfied by any finite transition system.

5 The cs-Paradigm

In the cs-paradigm we want to reason with arbitrary (Turing machine) computable Skolem functions, which allows us to overcome the restrictions of the up-paradigm. At first glance, this is a very ambitious goal, as it requires working with Turing machines processing infinite inputs and producing infinite outputs. We will apply the theory of uniformization of relations (a.k.a. synthesis) to approach this problem. Intuitively, in the uniformization problem one is given a relation $R \subseteq A \times B$ and the goal is to determine whether there is a function uniformizing R computed by a machine from some fixed class of machines, i.e., a partial function $f: A \rightarrow B$ such that $\text{dom}(f) = \text{dom}(R)$ and $\{(a, f(a)) \mid a \in \text{dom}(R)\} \subseteq R$. For the case where A is the Cartesian product of the set of traces of a transition system and B is the set of traces of the same transition system, we have captured the problem of computing a Skolem function as a uniformization problem.

Thus, for a sentence with quantifier prefix $\forall^*\exists^*$ we can compute computable Skolem functions by interpreting the problem as a uniformization problem. However, for more complex quantifier prefixes, this is no longer straightforward, as the dependencies between the variables have to be considered, e.g., the Skolem function of an existentially quantified variable only has inputs corresponding to outermore universally quantified variables, but may also depend on outermore existentially quantified variables, i.e., outputs are also (implicit) inputs for other functions.

Another issue is that Turing machines are a very expressive model of computation. Filiot and Winter [13] studied synthesis of computable functions from rational specifications (e.g., specifications recognized by a parity automaton): they proved that uniformization by Turing machines coincides with uniformization by transducers with bounded delay (if the domain of the specification is closed), a much nicer class of machines computing functions from infinite words to infinite words. Crucially, the functions computed by such transducers are also continuous in the Cantor topology (see refer to [13] for definitions and details). In the setting of Skolem functions for HyperLTL model-checking for $\forall^*\exists^*$ -sentences, that means that if two inputs agree on a “long” prefix, then the corresponding outputs also agree on a “long” prefix. Continuity is a desirable property in the context of the interactive simulation of Skolem functions described in the introduction: If Tracy has simulated a long prefix of the inputs for the Skolem functions, then future changes do not change the output prefixes already produced by the Skolem functions.

However, we will show that continuity also implies that not every HyperLTL sentence has computable (and therefore continuous) explanations. Thus, it is natural to ask whether it is decidable whether a given pair (\mathcal{T}, φ) has an explanation in the cs-paradigm. We prove that this is indeed the case for sentences with arbitrary quantifier prefixes.

We begin this section by reviewing the uniformization problem. Then, we introduce and investigate the cs-paradigm and provide an algorithm to determine the existence of computable Skolem functions.

5.1 Uniformization

In the following, for languages over the alphabet $\Sigma \times \Gamma$ recognized by parity automata, i.e., $L \subseteq (\Sigma \times \Gamma)^\omega$ we speak about its induced relation $R_L = \{(x, y) \in \Sigma^\omega \times \Gamma^\omega \mid \text{mrg}(x, y) \in L\}$. For the sake of readability, we often do not distinguish between (automata-recognizable) languages $L \subseteq (\Sigma \times \Gamma)^\omega$ and induced relations $R_L \subseteq \Sigma^\omega \times \Gamma^\omega$.

We say that a function $f: \Sigma^\omega \rightarrow \Gamma^\omega$ uniformizes a relation $R \subseteq \Sigma^\omega \times \Gamma^\omega$ if the domain of f is equal to the domain of R and the graph $\{(w, f(w)) \mid w \in \text{dom}(f)\}$ of f is a subset of R . The uniformization problem asks whether, for a given relation, there is a computable function that uniformizes it.

To define the computability of a function from Σ^ω to Γ^ω , we consider deterministic three-tape Turing machines \mathcal{M} with the following setup (following [13]): the first tape is a read-only, one-way tape and contains the input in Σ^ω , the second one is a two-way working tape, and the third one is a write-only, one-way tape on which the output in Γ^ω is generated. Formally, we say that \mathcal{M} computes the partial function $f: \Sigma^\omega \rightarrow \Gamma^\omega$ if, when started with input $w \in \text{dom}(f)$ on the first tape, \mathcal{M} produces (in the limit) the output $f(w)$ on the third tape. Note that we do not require the Turing machine to check whether its input is in the domain of f . We just require it to compute the correct output for those inputs in the domain, it may behave arbitrarily



Figure 2: The transition system for the proof of Theorem 2.

on inputs outside of the domain of f . This is done so that the uniformization function only has to capture the complexity of transforming possible inputs into outputs, but does not have to capture the complexity of checking whether an input is in the domain. In our setting, this can be taken care of by parity automata that can be effectively computed.

We say that such an \mathcal{M} has bounded delay, if there is a $d \in \mathbb{N}$ such that to compute the first n letters of the output only $n + d$ letters of the input are read (i.e., the other cells of the input tape are not visited before n output letters have been generated).

Remark 4. *If the domain of f is closed, then \mathcal{M} can be assumed (w.l.o.g.) to have bounded delay. This follows from the fact that the set Σ^ω is a compact space when equipped with the Cantor distance. A closed subset of a compact space is compact (see, e.g., [2]). Hence, $\text{dom}(f)$ is a compact space. Further, every computable function is continuous (see, e.g., [13]). Now, the Heine-Cantor theorem states that every continuous function between metric spaces $f: M \rightarrow N$ where M is a compact space is in fact uniformly continuous. Uniform continuity directly implies bounded delay.*

Now, we can formalize the relation between uniformization by computable functions (e.g., implemented by Turing machines) and uniformization by functions implementable by bounded-delay transducers (a much weaker machine model, informally introduced below, formally in Section 8) in the appendix, when considering relations with closed domain recognized by parity automata. Transducers extend (deterministic) automata with outputs on their transitions, and therefore implement functions of the form $f: \Sigma^\omega \rightarrow \Gamma^\omega$ by reading an input letter in Σ on each transition and producing a finite output word over Γ on each transition. We say a transducer has bounded delay if the length difference between read input and produced output is bounded for every prefix of a run.

Proposition 3 ([13]). *The following are equivalent for a relation R encoded by a parity automaton \mathcal{P} and with closed $\text{dom}(R)$:*

1. *R is uniformized by a computable function.*
2. *R is uniformized by a function implemented by a bounded-delay transducer.*

5.2 Computing Explanations in the cs-Paradigm

Our goal is to determine under which circumstances $\mathfrak{T} \models \varphi$ has an explanation in the cs-paradigm, i.e., there are computable Skolem functions witnessing $\mathfrak{T} \models \varphi$, and whether such Skolem functions can be computed by “simpler” models of computation, i.e., bounded-delay transducers. Intuitively, this aims to extend Proposition 3 from $\forall^* \exists^*$ sentences to arbitrary quantifier prefixes.

We start by establishing the following result, showing that the cs-paradigm is incomplete.

Theorem 2. *There is a HyperLTL sentence φ and a transition system \mathfrak{T} such that $\mathfrak{T} \models \varphi$ has no explanation in the cs-paradigm, i.e., it is not witnessed by computable Skolem functions.*

Proof. Consider the sentence $\varphi = \forall \pi \exists \pi'. (\mathbf{F} a_\pi) \leftrightarrow (\mathbf{X} a_{\pi'})$ and the transition system \mathfrak{T} in Figure 2 with $\text{Tr}(\mathfrak{T}) = \emptyset(2^{\{a\}})^\omega$.

Towards a contradiction, assume there is a computable Skolem function for π' . Then, due to Remark 4, there is also one that is implemented by a bounded-delay Turing machine \mathcal{M} , say with delay d . Now, let \mathcal{M} run on an input with prefix $\emptyset^{d+2} \in \text{Prfs}(\text{Tr}(\mathfrak{T}))$. As \mathcal{M} has bounded delay, it will produce the first two output letters $\emptyset A \in \emptyset 2^{\{a\}}$ after processing the prefix \emptyset^{d+2} (note that all traces of \mathfrak{T} start with \emptyset , the label of the initial state).

If $A = \emptyset$, then the output of \mathcal{M} on the input $\emptyset^{d+2}\{a\}^\omega$ starts with $\emptyset\emptyset$ (as this output only depends on the prefix \emptyset^{d+2}), but the input contains an $\{a\}$. These traces do not satisfy $(\mathbf{F} a_\pi) \leftrightarrow (\mathbf{X} a_{\pi'})$. On the other hand, if $A = \{a\}$, then the output of \mathcal{M} on the input \emptyset^ω starts with $\emptyset\{a\}$ (again, the output only depends on the prefix \emptyset^{d+2}), but the input contains no $\{a\}$. Again, these traces do not satisfy $(\mathbf{F} a_\pi) \leftrightarrow (\mathbf{X} a_{\pi'})$. So, in both cases, \mathcal{M} does not implement a Skolem function for π' , i.e., we have the desired contradiction. \square

So, as not every $\mathfrak{T} \models \varphi$ has an explanation in the cs-paradigm, it is natural to ask whether it is decidable whether, given \mathfrak{T} and φ , $\mathfrak{T} \models \varphi$ has an explanation in the cs-paradigm. Before we study this problem, we consider another example showing that for some transition system \mathfrak{T} and sentence φ , even if $\mathfrak{T} \models \varphi$ *does* have computable Skolem functions, not every (computable) Skolem function is a “good” Skolem function: Fixing a Skolem function for an outermore variable may block innermore variables having computable Skolem functions.

Example 3. Consider the sentence $\exists\pi\forall\pi'\exists\pi'' . (\mathbf{X} a_\pi) \rightarrow ((\mathbf{F} a_{\pi'}) \leftrightarrow (\mathbf{X} a_{\pi''}))$ and the transition system \mathfrak{T} from Figure 2. Recall that we have $\text{Tr}(\mathfrak{T}) = \emptyset(2^{\{a\}})^\omega$; especially, every trace of \mathfrak{T} starts with \emptyset . Also, as the quantification of π is not in the scope of any other quantifier we can identify Skolem functions for π with traces that are assigned to π .

Now, if we pick a trace t for π with $a \in t(1)$ then there is no computable Skolem function for π'' (see Theorem 2). However, if we pick a trace t for π with $a \notin t(1)$ then every function is a Skolem function for π'' , as satisfaction is independent of the choices for π' and π'' in this case. In particular, π'' has a computable Skolem function.

Thus, the wrong choice of a (computable) Skolem function for some variable may result in other variables not having computable Skolem functions. By carefully accounting for the dependencies between the Skolem functions we show that the existence of computable Skolem functions is decidable.

Theorem 3. The following problem is decidable: “Given a transition system \mathfrak{T} and a HyperLTL sentence φ with $\mathfrak{T} \models \varphi$, is $\mathfrak{T} \models \varphi$ witnessed by computable Skolem functions?”, i.e., does $\mathfrak{T} \models \varphi$ have an explanation in the cs-paradigm? If the answer is yes, our algorithm computes bounded-delay transducers implementing such Skolem functions.

To show the result, we characterize the existence of computable Skolem functions by a game, generalizing previous work on uniformization via two-player games with delay [24, 13] (which covers essentially the special case of $\forall^*\exists^*$ -sentences) to multi-player games (one player for each existentially quantified variable) of imperfect information [4] (which captures the fact that an existentially quantified variable may only depend on variables that are quantified outermore). The full proof is presented in the appendix.

Hence, while the cs-paradigm is not complete, one can at least decide whether $\mathfrak{T} \models \varphi$ has an explanation in this paradigm, and, if yes, effectively compute such an explanation.

6 Related Work

As explained in the introduction, computing counterexamples for debugging is the most important application of model-checking. In the framework of LTL model-checking, a counterexample is a single trace of the system that violates the specification. Such a counterexample is typically obtained by running a model-checking algorithm (which are in fact based on searching for such counterexamples). Variations of the problem include bounded model-checking [6], which searches for “short” counterexamples. Also, counterexample-guided abstraction refinement [7] and bounded synthesis [16] rely on counterexample computation.

Counterexamples have not only been studied in the realm of linear-time logics, but also for many other frameworks, e.g., for \forall CTL [8], for CTL [19, 31], for probabilistic temporal logics [12, 20], and for discrete-time Markov models [1].

Finally, counterexamples have also been studied in the realm of HyperLTL model-checking. Horak et al. [22] developed HYPERVIS, a webtool which provides interactive visualizations of a given model, specification, and counterexample computed by the HyperLTL model-checker MCHYPER [15]. In complementary

work, Coenen et al. [11] present a causality-based explanation method for HyperLTL counterexamples, again computed by MCHYPER. However, the counterexamples computed by MCHYPER are just sets of traces that violate the formula. More specifically, MCHYPER only considers the outermost universal quantifiers and returns a variable assignment to those. This is obviously complete for formulas with quantifier-prefix \forall^* , i.e., without existential quantifiers, but not for more general formulas. In fact, this approach ignores the dynamic dependencies between universally and existentially quantified variables that is captured by Skolem functions, which we analyze here. Finally, let us also mention that counterexamples are the foundation of the bounded synthesis algorithm for \forall^* HyperLTL specifications [14]. In this setting, it is again sufficient to only consider sets of traces and not general Skolem functions.

7 Conclusion

We have presented two paradigms to explain why a given transition system satisfies a given HyperLTL formula or, equivalently, to provide counterexamples in case the system does not satisfy the formula. The up-paradigm is complete (if $\mathfrak{T} \models \varphi$, then this has an explanation), but restricted to ultimately periodic traces. The cs-paradigm handles arbitrary computable explanations, but is incomplete: $\mathfrak{T} \models \varphi$ may not have a computable explanation. However, we have shown that the existence of computable explanations is decidable, and that they are effectively computable (whenever they exist).

After having laid the theoretical foundations, it remains to investigate under which circumstances such explanations can be useful in the verification work-flow. For a restricted setting, this line of work has been studied as discussed in Section 6. We propose to continue this line of work for more expressive fragments of HyperLTL and explanations in both the up- and cs-paradigm. On the theoretical side, in future work we will study the size of transducers implementing Skolem functions and the complexity of deciding the existence of computable Skolem functions. In general, this is infeasible, as HyperLTL model-checking is already TOWER-complete, but for fragments with few quantifier alternations, the problem is simpler. For example, we expect that results on delay games with LTL winning conditions [25] can be adapted to show that deciding the existence of computable Skolem functions for $\forall^*\exists^*$ -sentences is 3EXPTIME-complete.

Finally, we propose to compare our work to the HyperLTL model-checking algorithm for $\forall^*\exists^*$ -sentences of Beutner and Finkbeiner [5], which relies on a delay-free game extended with prophecies that require the player in charge of the universal variable to make commitments about future moves (w.r.t. to membership in a finite list of ω -regular properties). In particular, to the best of our knowledge, this approach has not been extended beyond the $\forall^*\exists^*$ -fragment.

Epilogue. After her lunch break, Tracy walks by the printer room and finds a second copy of her document in the printer tray. It turns out the print system satisfies φ_{id} , but the explanation shows that there is a large lag in the network.

References

- [1] Erika Ábrahám, Bernd Becker, Christian Dehnert, Nils Jansen, Joost-Pieter Katoen, and Ralf Wimmer. Counterexample generation for discrete-time Markov models: An introductory survey. In Marco Bernardo, Ferruccio Damiani, Reiner Hähnle, Einar Broch Johnsen, and Ina Schaefer, editors, *SFM 2014*, volume 8483 of *LNCS*, pages 65–121. Springer, 2014.
- [2] A. V. Arkhangel’skiĭ and Vitaly V. Fedorchuk. The basic concepts and constructions of general topology. In L. S. Arkhangel’skiĭ, A. V. and Pontryagin, editor, *General Topology I: Basic Concepts and Constructions Dimension Theory*, pages 1–90. Springer, 1990.
- [3] Alejandro Barredo Arrieta, Natalia Díaz Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila,

- and Francisco Herrera. Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI. *Inf. Fusion*, 58:82–115, 2020.
- [4] Dietmar Berwanger, Anup Basil Mathew, and Marie van den Bogaard. Hierarchical information and the synthesis of distributed strategies. *Acta Informatica*, 55(8):669–701, 2018.
 - [5] Raven Beutner and Bernd Finkbeiner. Prophecy variables for hyperproperty verification. In *CSF 2022*, pages 471–485. IEEE, 2022.
 - [6] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Adv. Comput.*, 58:117–148, 2003.
 - [7] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In E. Allen Emerson and A. Prasad Sistla, editors, *CAV*, volume 1855 of *LNCS*, pages 154–169. Springer, 2000.
 - [8] Edmund M. Clarke, Somesh Jha, Yuan Lu, and Helmut Veith. Tree-like counterexamples in model checking. In *LICS 2002*, pages 19–29. IEEE Computer Society, 2002.
 - [9] Michael R. Clarkson, Bernd Finkbeiner, Masoud Kolehini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In Martín Abadi and Steve Kremer, editors, *POST 2014*, volume 8414 of *LNCS*, pages 265–284. Springer, 2014.
 - [10] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *J. Comput. Secur.*, 18(6):1157–1210, 2010.
 - [11] Norine Coenen, Raimund Dachsel, Bernd Finkbeiner, Hadar Frenkel, Christopher Hahn, Tom Horak, Niklas Metzger, and Julian Siber. Explaining hyperproperty violations. In Sharon Shoham and Yakir Vizel, editors, *CAV 2022, Part I*, volume 13371 of *LNCS*, pages 407–429. Springer, 2022.
 - [12] Berteun Damman, Tingting Han, and Joost-Pieter Katoen. Regular expressions for PCTL counterexamples. In *QEST 2008*, pages 179–188. IEEE Computer Society, 2008.
 - [13] Emmanuel Filiot and Sarah Winter. Synthesizing computable functions from rational specifications over infinite words. *Int. J. Found. Comput. Sci.*, 35(01n02):179–214, 2024.
 - [14] Bernd Finkbeiner, Christopher Hahn, Philip Lukert, Marvin Stenger, and Leander Tentrup. Synthesis from hyperproperties. *Acta Informatica*, 57(1-2):137–163, 2020.
 - [15] Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for Model Checking HyperLTL and HyperCTL*. In Daniel Kroening and Corina S. Pasareanu, editors, *CAV 2015, Part I*, volume 9206 of *LNCS*, pages 30–48. Springer, 2015.
 - [16] Bernd Finkbeiner and Sven Schewe. Bounded synthesis. *Int. J. Softw. Tools Technol. Transf.*, 15(5-6):519–539, 2013.
 - [17] Bernd Finkbeiner and Martin Zimmermann. The First-Order Logic of Hyperproperties. In *STACS 2017*, volume 66 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.
 - [18] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *LNCS*. Springer, 2002.
 - [19] Arie Gurfinkel and Marsha Chechik. Proof-like counter-examples. In Hubert Garavel and John Hatcliff, editors, *TACAS 2003*, volume 2619 of *LNCS*, pages 160–175. Springer, 2003.
 - [20] Tingting Han and Joost-Pieter Katoen. Counterexamples in probabilistic model checking. In Orna Grumberg and Michael Huth, editors, *TACAS 2007*, volume 4424 of *LNCS*, pages 72–86. Springer, 2007.

- [21] Michael Holtmann, Lukasz Kaiser, and Wolfgang Thomas. Degrees of lookahead in regular infinite games. *Log. Methods Comput. Sci.*, 8(3), 2012.
- [22] Tom Horak, Norine Coenen, Niklas Metzger, Christopher Hahn, Tamara Flemisch, Julián Méndez, Dennis Dimov, Bernd Finkbeiner, and Raimund Dachsel. Visual analysis of hyperproperties for understanding model checking results. *IEEE Trans. Vis. Comput. Graph.*, 28(1):357–367, 2022.
- [23] Frederick A. Hosch and Lawrence H. Landweber. Finite delay solutions for sequential conditions. In *ICALP 1972*, pages 45–60, 1972.
- [24] Felix Klein and Martin Zimmermann. How much lookahead is needed to win infinite games? *Log. Methods Comput. Sci.*, 12(3), 2016.
- [25] Felix Klein and Martin Zimmermann. Prompt delay. In Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen, editors, *FSTTCS 2016*, volume 65 of *LIPICs*, pages 43:1–43:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
- [26] Robert P. Kurshan. Verification technology transfer. In Orna Grumberg and Helmut Veith, editors, *25 Years of Model Checking - History, Achievements, Perspectives*, volume 5000 of *LNCs*, pages 46–64. Springer, 2008.
- [27] Corto Mascle and Martin Zimmermann. The keys to decidable HyperLTL satisfiability: Small models or very simple formulas. In Maribel Fernández and Anca Muscholl, editors, *CSL 2020*, volume 152 of *LIPICs*, pages 29:1–29:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- [28] Amir Pnueli. The temporal logic of programs. In *FOCS 1977*, pages 46–57. IEEE, Oct 1977.
- [29] Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *FOCS 1990, Volume II*, pages 746–757. IEEE Computer Society, 1990.
- [30] Markus N. Rabe. *A temporal logic approach to information-flow control*. PhD thesis, Saarland University, 2016.
- [31] Sharon Shoham and Orna Grumberg. A game-based framework for CTL counterexamples and 3-valued abstraction-refinement. *ACM Trans. Comput. Log.*, 9(1):1, 2007.

Appendix

8 Transducers

A (one-way deterministic finite) transducer \mathcal{T} is a tuple $(Q, \Sigma, \Gamma, q_I, \delta, \Omega)$ that consists of a finite set Q of states containing the initial state q_I , an input alphabet Σ , an output alphabet Γ , a transition function $\delta: Q \times \Sigma \rightarrow Q \times \Gamma^*$, and a coloring $\Omega: Q \rightarrow \mathbb{N}$. The (unique) run of \mathcal{T} on an input $w = w(0)w(1)w(2) \cdots \in \Sigma^\omega$ is the sequence $q_0q_1q_2 \cdots$ of states defined by $q_0 = q_I$ and q_{i+1} being the unique state with $\delta(q_i, w(i)) = (q_{i+1}, x_i)$ for some $x_i \in \Gamma^*$. The run is accepting if the maximal color appearing infinitely often in $\Omega(q_0)\Omega(q_1)\Omega(q_2) \cdots$ is even. With the run $q_0q_1q_2 \cdots$ on w we associate the output $x_0x_1x_2 \cdots$, where the x_i are as defined above. As the transducer is deterministic, it induces a map from inputs to outputs. Note that the output may, a priori, be a finite or an infinite word over Γ . In the following, we only consider transducers where the output is infinite for every input with an accepting run. In this case, \mathcal{T} computes a partial function $f_{\mathcal{T}}: \Sigma^\omega \rightarrow \Gamma^\omega$ defined as follows: the domain of $f_{\mathcal{T}}$ is the set of infinite words $w \in \Sigma^\omega$ such that the run of \mathcal{T} on w is accepting and $f_{\mathcal{T}}(w)$ is the output induced by this (unique) run.

We say that \mathcal{T} has delay $d \in \mathbb{N}$ if for every accepting run and every induced sequence $x_0x_1x_2 \cdots$ of outputs (x_i is the output on the i -th transition), we have $i - d \leq |x_0 \cdots x_{i-1}| \leq i$ for all $i \geq 0$, i.e., the output is, at any moment during an accepting run, at most d letters shorter than the input and never longer. We say that \mathcal{T} is a bounded-delay transducer if there is a d such that it has delay d .

9 Proof of Theorem 3

This section is devoted to the proof of Theorem 3, i.e., we prove that it is decidable whether, for a given transition system \mathfrak{T} and HyperLTL sentence φ with $\mathfrak{T} \models \varphi$, $\mathfrak{T} \models \varphi$ is witnessed by computable Skolem functions. Furthermore, if the answer is yes, then we show how to compute such Skolem functions (represented by bounded-delay transducers).

We begin by giving some intuition for the coming proof by first looking at the special case of a sentence of the form $\forall\pi\exists\pi'. \psi$ where ψ is quantifier-free. Then, we need to decide whether there is a computable function $f: \text{Tr}(\mathfrak{T}) \rightarrow \text{Tr}(\mathfrak{T})$ such that $\{\pi \mapsto t, \pi' \mapsto f(t)\} \models \psi$. Note that

$$\{\text{mrg}(t, t') \mid t, t' \in \text{Tr}(\mathfrak{T}) \text{ and } \{\pi \mapsto t, \pi' \mapsto t'\} \models \psi\}$$

is accepted by a parity automaton (see Proposition 2). Hence, the problem boils down to a uniformization problem for an ω -regular relation (recall Subsection 5.1). This problem was first posed (and partially solved) by Hosch and Landweber in 1971 [23] and later completely solved in a series of works [21, 24, 13]. Let us sketch the main ideas underlying the solution, as we will generalize them in the following.

Let $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$ be ω -regular. Then, the existence of a computable function $f: \Sigma_I^\omega \rightarrow \Sigma_O^\omega$ that uniformizes L is captured by a (perfect information) two-player game $\Gamma(L)$ of infinite duration played between Player I (the input player) and Player O (the output player) in rounds $r = 0, 1, 2, \dots$ as follows: In each round r , Player I picks an $a_r \in \Sigma_I$ and then Player O picks a $b_r \in \Sigma_O \cup \{\varepsilon\}$. Thus, the outcome of a play of $\Gamma(L)$ is an infinite word $a_0a_1a_2\cdots \in \Sigma_I^\omega$ picked by Player I and a finite or infinite word $b_0b_1b_2\cdots \in \Sigma_O^* \cup \Sigma_O^\omega$ picked by Player O . The outcome is winning for Player O if $a_0a_1a_2\cdots \in \text{dom}(L)$ implies $\text{mrg}(a_0a_1a_2\cdots, b_0b_1b_2\cdots) \in L$ (which requires that $b_0b_1b_2\cdots$ is infinite, i.e., Player O has to pick infinitely often a letter from Σ_O). Now, one can show that a winning strategy for Player O can be turned into a function that uniformizes L and every function uniformizing L can be turned into a winning strategy for Player O .

So far, the uniformizing function may be arbitrary, in particular not computable. Also, the delay between input and output in plays that are consistent with a strategy can be unbounded, e.g., if Player O picks ε in every second round. A crucial insight is that this is not necessary: for every ω -regular $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$ such that Player O wins $\Gamma(L)$, there is a bound ℓ (that only depends on the size of a (minimal) parity automaton accepting L) such that she has a winning strategy that picks ε at most ℓ times [21].

This insight allows to change the rules of $\Gamma(L)$, giving Player O the advantage gained by using ε a bounded number of times from the beginning of a play and grouping moves into blocks of letters of a fixed length. How the block length is obtained is explained below.

The block game $\Gamma_b(L)$ is played in rounds $r = 0, 1, 2, \dots$ as follows: In round 0, Player I picks two blocks $x_0, x_1 \in \Sigma_I^\ell$ and then Player O picks a block $y_0 \in \Sigma_O^\ell$. Then, in every round $r > 0$, Player I picks a block $x_{r+1} \in \Sigma_I^\ell$ and then Player O picks a block $y_r \in \Sigma_O^\ell$. Note that Player I is one block ahead, as he has to pick two blocks in round 0. This accounts for the delay allowed in the definition of computable functions. The outcome of a play of $\Gamma_b(L)$ is an infinite word $x_0x_1x_2\cdots \in \Sigma_I^\omega$ picked by Player I and an infinite word $y_0y_1y_2\cdots \in \Sigma_O^\omega$ picked by Player O . The outcome is winning for Player O if $x_0x_1x_2\cdots \in \text{dom}(L)$ implies $\text{mrg}(x_0x_1x_2\cdots, y_0y_1y_2\cdots) \in L$.

Now, one can show that L is uniformizable if and only if Player O has a winning strategy for $\Gamma_b(L)$. As $\Gamma_b(L)$ is a finite two-player game with ω -regular winning condition, Player O has a finite-state winning strategy (one implemented by a transducer). Such a finite-state winning strategy can be turned into a computable function uniformizing L , as a transducer can be simulated by a Turing machine. Hence, $\Gamma_b(L)$ does indeed characterize uniformizability of ω -regular relations by computable functions.

Next, let us give some intuition how to obtain the bound ℓ . To this end, let \mathcal{P} be an automaton over some alphabet $\Sigma_I \times \Sigma_O$ (we ignore the acceptance condition in this discussion and later hint at how this is taken into account). As usual, if two finite words $w_0 \in (\Sigma_I \times \Sigma_O)^*$ and $w_1 \in (\Sigma_I \times \Sigma_O)^*$ induce the same state transformations (e.g., for all states p and q , processing w_0 from p leads \mathcal{P} to q if and only if processing w_1 from p leads \mathcal{P} to q), then these words are indistinguishable for \mathcal{P} (again, we are ignoring acceptance for the time being), i.e., one can replace w_0 by w_1 without changing the possible runs that \mathcal{P} has. This indistinguishability is captured by an equivalence relation over $(\Sigma_I \times \Sigma_O)^*$ with finite index.

However, to capture the interaction described in $\Gamma(L)$ above, we need a more refined approach. Assume Player I picks a sequence $x \in \Sigma_I^*$ of letters. Then, Player O will have to “complete” this block by picking a block $y \in \Sigma_O^{|x|}$ so that $\text{mrg}(x, y)$ is processed by the automaton. In this situation, we can say that x_0 and $x_1 \in \Sigma_I^*$ are equivalent if they are indistinguishable w.r.t. to their completions to words of the form $\text{mrg}(x_i, y_i)$, e.g., for all states p and q , there is a completion $\text{mrg}(x_0, y_0) \in (\Sigma_I \times \Sigma_O)^*$ of x_0 that leads \mathcal{P} from p to q if and only if there is a completion $\text{mrg}(x_1, y_1) \in (\Sigma_I \times \Sigma_O)^*$ of x_1 that leads \mathcal{P} from p to q . Intuitively, one does not need to distinguish between x_0 and x_1 because they allow Player O to achieve the same state transformations in \mathcal{P} . This indistinguishability is captured by an equivalence relation over Σ_I^* of finite index. Now, ℓ can be picked as an upper bound on the length of a minimal word in all equivalence classes.

Thus, the intuition behind the definition of $\Gamma_b(L)$ is that blocks of length ℓ are *rich* enough to capture the full strategic choices for both players in $\Gamma(L)$: every longer word has an equivalent one of length at most ℓ .

Finally, let us briefly mention how to deal with the acceptance condition we have ignored thus far. As the state transformations are concerned with finite runs of a parity automaton, we can just keep track of the maximal color occurring during this run, all other information is irrelevant. Keeping track of the color can be achieved using the states of the automaton.

After having sketched the special case of a sentence of the form $\forall\pi\exists\pi'. \psi$, let us now illustrate the challenges we have to address to deal with more quantifier alternations, e.g., for a sentence of the form $\varphi = \forall\pi_0\exists\pi_1\cdots\forall\pi_{k-2}\exists\pi_{k-1}. \psi$.

- We will consider a multi-player game with one player being in charge of providing traces for the universally quantified variables (generalizing Player I above) and one variable player for each existentially quantified variable (generalizing Player O above), i.e., altogether we have $\frac{k}{2} + 1$ players. Thus, the player in charge of the universally quantified variables produces traces t_0, t_2, \dots, t_{k-2} while each variable player produces a trace t_i (one for each odd i). These traces are again picked block-wise in rounds.
- The choices by the variable player producing t_i (i.e., i is odd) may only depend on the traces t_0, t_1, \dots, t_{i-1} in order to faithfully capture the semantics of φ . Hence, we need to consider a game of imperfect information, which allows us to hide the traces t_{i+1}, \dots, t_{k-1} from the player in charge of π_i .
- Recall that Player I is always one block ahead of Player O in $\Gamma_b(L)$, which accounts for the delay allowed in the definition of computable functions. With k traces to be picked (and t_i depending on t_0, t_1, \dots, t_{i-1}), there must be a gap of one block for each even i .

In the following, we present a game that captures the existence of computable Skolem functions and then show that it can effectively be solved. As a byproduct, we obtain Skolem functions implemented by bounded-delay transducers whenever computable Skolem functions exist. We begin by defining the appropriate equivalence relations in Subsection 9.1 and then use them to define our game-theoretic characterization. To this end, we first introduce an abstract game in Subsection 9.2 to prove the characterization correct, and then formalize the abstract game as a multi-player game of imperfect information in Subsection 9.4 to prove decidability of our characterization. Before that, we introduce multi-player games of imperfect information in Subsection 9.3.

Throughout the remainder of this section, we fix a HyperLTL sentence φ and a transition system \mathfrak{T} with $\mathfrak{T} \models \varphi$. We assume (w.l.o.g.)⁴ $\varphi = \forall\pi_0\exists\pi_1\cdots\forall\pi_{k-2}\exists\pi_{k-1}. \psi$, define $\varphi_i = Q_i\pi_i Q_{i+1}\pi_{i+1}\cdots\forall\pi_{k-2}\exists\pi_{k-1}. \psi$ for $i \in \{0, 1, \dots, k-1\}$ and $\varphi_k = \psi$, and use the automata $\mathcal{P}_i^{\mathfrak{T}} = (Q_i, (2^{\text{AP}})^i, q_{I,i}, \delta_i, \Omega_i)$ constructed in Remark 3 satisfying

$$L(\mathcal{P}_i^{\mathfrak{T}}) = \{\text{mrg}(\Pi(\pi_0), \dots, \Pi(\pi_{i-1})) \mid \Pi(\pi_j) \in \text{Tr}(\mathfrak{T}) \text{ for all } 0 \leq j < i \text{ and } (\text{Tr}(\mathfrak{T}), \Pi) \models \varphi_i\}.$$

Recall that the $\mathcal{P}_i^{\mathfrak{T}}$ are all deterministic, which will be crucial in the following.

⁴The following reasoning can easily be extended to general sentences with arbitrary quantifier prefixes, albeit at the cost of more complex notation.

9.1 The Equivalence Relations

We write $\mathcal{P}: p \xrightarrow{w} q$ for states p, q of a parity automaton \mathcal{P} , if \mathcal{P} (over an alphabet Σ) has a run from p to q processing the word $w \in \Sigma^*$. Similarly, we write $\mathfrak{T}: v \xrightarrow{w} v'$ for vertices v, v' of a transition system \mathfrak{T} , if \mathfrak{T} has a path from v to v' labeled by the word $w \in (2^{\text{AP}})^*$. Also, we need to work with tuples of finite words

of the same length. To simplify our notation, from now on we only write $\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{i-1} \end{pmatrix}$ if each w_j is a word in $(2^{\text{AP}})^*$ such that $|w_0| = |w_1| = \dots = |w_{i-1}|$.

Note that our notation $\mathcal{P}: p \xrightarrow{w} q$ expressing the existence of a run does not keep track of the colors encountered during the run. This is because we modify the automata $\mathcal{P}_i^{\mathfrak{T}}$ to keep track of the maximal color encountered along a finite run using their state space, as motivated above.

Remark 5. Let $C \subseteq \mathbb{N}$ be finite, let $\alpha = \alpha_0 \alpha_1 \alpha_1 \dots \in C^\omega$ where each α_i is a nonempty finite word (over C) and define c_i to be the maximal color occurring in α_i . Then, α satisfies the parity condition if and only if $c_0 c_1 c_2 \dots$ satisfies the parity condition.

To exploit this, we define the color-tracking automaton

$$\mathcal{C}_i = (Q_i \times \Omega_i(Q_i), (2^{\text{AP}})^i, (q_{I,i}, \Omega_i(q_{I,i})), \delta'_i, \Omega'_i)$$

where

- $\delta'_i((q, c), a) = (q', \max\{c, \Omega_i(q')\})$ where $q' = \delta_i(q, a)$ and
- $\Omega'_i(q, c) = c$.

Note that \mathcal{C}_i is deterministic, as $\mathcal{P}_i^{\mathfrak{T}}$ is deterministic. In the following, we do *not* care about the language recognized by \mathcal{C}_i (which in fact is not necessarily equal to $L(\mathcal{P}_i^{\mathfrak{T}})$), but only about finite runs in \mathcal{C}_i . The following remark formalizes how \mathcal{C}_i keeps track of the maximal color of runs of $\mathcal{P}_i^{\mathfrak{T}}$, where the second item relies on the fact that $\mathcal{P}_i^{\mathfrak{T}}$ is deterministic and on Remark 5.

Remark 6.

1. For all finite words w (over the alphabet of $\mathcal{P}_i^{\mathfrak{T}}$) and all states p, q and colors c of $\mathcal{P}_i^{\mathfrak{T}}$: $\mathcal{C}_i: (p, \Omega_i(p)) \xrightarrow{w} (q, c)$ if and only if $\mathcal{P}_i^{\mathfrak{T}}$ has a run from p to q processing w with maximal color c .
2. Consequently, for all infinite sequences w_0, w_1, w_2, \dots of nonempty finite words and all infinite sequences $(q_0, c_0), (q_1, c_1), (q_2, c_2), \dots$ of states of \mathcal{C}_i such that

- $\mathcal{C}_i: (q_{I,i}, \Omega_i(q_{I,i})) \xrightarrow{w_0} (q_0, c_0)$ and
- $\mathcal{C}_i: (q_{n-1}, \Omega_i(q_{n-1})) \xrightarrow{w_n} (q_n, c_n)$ for all $n > 0$:

$w_0 w_1 w_2 \dots \in L(\mathcal{P}_i^{\mathfrak{T}})$ if and only if $c_0 c_1 c_2 \dots = \Omega'_i(q_0, c_0) \Omega'_i(q_1, c_1) \Omega'_i(q_2, c_2) \dots$ satisfies the parity condition.

We continue by defining an equivalence relation \equiv_i between $(i-1)$ -tuples of (finite) words with the intuition that two such tuples of words are i -equivalent if they both can be “completed” with an i -th component such that the completed i -tuples induce the same state transformations in \mathcal{C}_i , i.e., such tuples do not need to be distinguished in the following.

Formally, fix some even $i \in \{2, 4, \dots, k\}$, some $\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{i-2} \end{pmatrix}$, as well as states p, q of \mathcal{C}_i . We say that $w_{i-1} \in$

$(2^{\text{AP}})^{|w_0|}$ is a (p, q) -completion of $\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{i-2} \end{pmatrix}$ if $\mathcal{C}_i: p \xrightarrow{\text{mrg}(w_0, w_1, \dots, w_{i-1})} q$. We define $\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{i-2} \end{pmatrix} \equiv_i \begin{pmatrix} \tilde{w}_0 \\ \tilde{w}_1 \\ \vdots \\ \tilde{w}_{i-2} \end{pmatrix}$ if

and only if

- for all states p, q of \mathcal{C}_i : $\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{i-2} \end{pmatrix}$ has a (p, q) -completion if and only if $\begin{pmatrix} \tilde{w}_0 \\ \tilde{w}_1 \\ \vdots \\ \tilde{w}_{i-2} \end{pmatrix}$ has a (p, q) -completion,

and

- for all vertices u, v of \mathfrak{T} and all $j \in \{0, 1, \dots, i-2\}$: $\mathfrak{T}: u \xrightarrow{w_j} v$ if and only if $\mathfrak{T}: u \xrightarrow{\tilde{w}_j} v$.

Let us collect some facts about the \equiv_i . Here, the first item is straightforward while the second follows from the second item of the definition of \equiv_i for $i = k$.

Remark 7.

1. Every \equiv_i is an equivalence relation of finite index.

2. Let $\begin{pmatrix} w_0^0 \\ w_1^0 \\ \vdots \\ w_{k-2}^0 \end{pmatrix}, \begin{pmatrix} w_0^1 \\ w_1^1 \\ \vdots \\ w_{k-2}^1 \end{pmatrix}, \begin{pmatrix} w_0^2 \\ w_1^2 \\ \vdots \\ w_{k-2}^2 \end{pmatrix}, \dots$ and $\begin{pmatrix} \tilde{w}_0^0 \\ \tilde{w}_1^0 \\ \vdots \\ \tilde{w}_{k-2}^0 \end{pmatrix}, \begin{pmatrix} \tilde{w}_0^1 \\ \tilde{w}_1^1 \\ \vdots \\ \tilde{w}_{k-2}^1 \end{pmatrix}, \begin{pmatrix} \tilde{w}_0^2 \\ \tilde{w}_1^2 \\ \vdots \\ \tilde{w}_{k-2}^2 \end{pmatrix}, \dots$ be two sequences of tuples of words such that $\begin{pmatrix} w_0^n \\ w_1^n \\ \vdots \\ w_{k-2}^n \end{pmatrix} \equiv_k \begin{pmatrix} \tilde{w}_0^n \\ \tilde{w}_1^n \\ \vdots \\ \tilde{w}_{k-2}^n \end{pmatrix}$ for all $n \in \mathbb{N}$, and let $i \in \{0, 1, \dots, k-2\}$. Then, $w_i^0 w_i^1 w_i^2 \dots \in \text{Tr}(\mathfrak{T})$ if and only if $\tilde{w}_i^0 \tilde{w}_i^1 \tilde{w}_i^2 \dots \in \text{Tr}(\mathfrak{T})$.

However, \equiv_i is still too coarse, as, for example, w_0 and \tilde{w}_0 can be \equiv_2 -equivalent, but still have (p, q) -completions w_1 and \tilde{w}_1 respectively, such that $\begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$ and $\begin{pmatrix} \tilde{w}_0 \\ \tilde{w}_1 \\ \tilde{w}_2 \end{pmatrix}$ are not \equiv_4 -equivalent for any w_2 and \tilde{w}_2 . Hence, (intuitively) we need to refine \equiv_2 with respect to the equivalence classes of all possible completions.

Formally, we inductively define a refined equivalence relation \equiv_i^{rf} over words of the form $\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{i-2} \end{pmatrix}$ for

every $i \in \{2, 3, \dots, k\}$ (i.e., also for the odd i !).

- We define \equiv_k^{rf} to be equal to \equiv_k .

- For $i \in \{3, 5, \dots, k-1\}$, we define $\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{i-2} \end{pmatrix} \equiv_i^{\text{rf}} \begin{pmatrix} \tilde{w}_0 \\ \tilde{w}_1 \\ \vdots \\ \tilde{w}_{i-2} \end{pmatrix}$ if and only if

- for all $w_{i-1} \in (2^{\text{AP}})^{|w_0|}$ there exists $\tilde{w}_{i-1} \in (2^{\text{AP}})^{|\tilde{w}_0|}$ such that $\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{i-1} \end{pmatrix} \equiv_{i+1}^{\text{rf}} \begin{pmatrix} \tilde{w}_0 \\ \tilde{w}_1 \\ \vdots \\ \tilde{w}_{i-1} \end{pmatrix}$, and
- for all $\tilde{w}_{i-1} \in (2^{\text{AP}})^{|\tilde{w}_0|}$ there exists $w_{i-1} \in (2^{\text{AP}})^{|w_0|}$ such that $\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{i-1} \end{pmatrix} \equiv_{i+1}^{\text{rf}} \begin{pmatrix} \tilde{w}_0 \\ \tilde{w}_1 \\ \vdots \\ \tilde{w}_{i-1} \end{pmatrix}$.
- For $i \in \{2, 4, \dots, k-2\}$, define $\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{i-2} \end{pmatrix} \equiv_i^{\text{rf}} \begin{pmatrix} \tilde{w}_0 \\ \tilde{w}_1 \\ \vdots \\ \tilde{w}_{i-2} \end{pmatrix}$ if and only if $\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{i-2} \end{pmatrix} \equiv_i \begin{pmatrix} \tilde{w}_0 \\ \tilde{w}_1 \\ \vdots \\ \tilde{w}_{i-2} \end{pmatrix}$ and if for all states p, q of \mathcal{C}_i ,
 - for all (p, q) -completions w_{i-1} of $\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{i-2} \end{pmatrix}$ there exists a (p, q) -completion \tilde{w}_{i-1} of $\begin{pmatrix} \tilde{w}_0 \\ \tilde{w}_1 \\ \vdots \\ \tilde{w}_{i-2} \end{pmatrix}$ such that $\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{i-1} \end{pmatrix} \equiv_{i+1}^{\text{rf}} \begin{pmatrix} \tilde{w}_0 \\ \tilde{w}_1 \\ \vdots \\ \tilde{w}_{i-1} \end{pmatrix}$, and
 - for all (p, q) -completions \tilde{w}_{i-1} of $\begin{pmatrix} \tilde{w}_0 \\ \tilde{w}_1 \\ \vdots \\ \tilde{w}_{i-2} \end{pmatrix}$ there exists a (p, q) -completion w_{i-1} of $\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{i-2} \end{pmatrix}$ such that $\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{i-1} \end{pmatrix} \equiv_{i+1}^{\text{rf}} \begin{pmatrix} \tilde{w}_0 \\ \tilde{w}_1 \\ \vdots \\ \tilde{w}_{i-1} \end{pmatrix}$.

Lemma 1.

1. Every \equiv_i^{rf} is an equivalence relation of finite index.
2. For even i , \equiv_i^{rf} refines \equiv_i .

Proof. The second item follows from the definition of \equiv_i^{rf} , so we only consider the first one.

To begin, recall that \equiv_k^{rf} is defined as \equiv_k . So, the result follows from Remark 7. For $i < k$, note that both cases in the definition of \equiv_i^{rf} share some similarities, i.e., the pattern of quantification is

$$\forall w_{i-1} \exists \tilde{w}_{i-1} [\dots] \quad \text{and} \quad \forall \tilde{w}_{i-1} \exists w_{i-1} [\dots]$$

(we ignore the additional quantification over states of \mathcal{C}_i in the case of an even i for the time being). Hence, we present an abstract proof that captures the essence of both cases simultaneously.

Fix some set $S \times E$, let \equiv be an equivalence relation over $S \times E$ of finite index, and let $R \subseteq S \times E$ be a relation. Furthermore, let \equiv' be the relation over S defined as $s \equiv' \tilde{s}$ if and only if

- for all $e \in E$ such that $(s, e) \in R$ there exists an $\tilde{e} \in E$ such that $(\tilde{s}, \tilde{e}) \in R$ and $\begin{pmatrix} s \\ e \end{pmatrix} \equiv \begin{pmatrix} \tilde{s} \\ \tilde{e} \end{pmatrix}$, and

- for all $\tilde{e} \in E$ such that $(\tilde{s}, \tilde{e}) \in R$ there exists an $e \in E$ such that $(s, e) \in R$ and $\begin{pmatrix} s \\ e \end{pmatrix} \equiv \begin{pmatrix} \tilde{s} \\ \tilde{e} \end{pmatrix}$.

It is straightforward to verify that \equiv' is an equivalence relation, so let us focus on the index of \equiv' .

We define $ext(s)$ for $s \in S$ to be the set of \equiv equivalence classes of tuples of the form $\begin{pmatrix} s \\ e \end{pmatrix}$, where e ranges over elements from E with $(s, e) \in R$. Now, we define $s \equiv_{ext} \tilde{s}$ if and only if $ext(s) = ext(\tilde{s})$, which is an equivalence relation of finite index: The codomain of ext has at most 2^n elements, where n is the index of \equiv . Finally, \equiv_{ext} refines \equiv' , which implies that \equiv' has finite index as well.

Now, our result follows by induction over i from $k-1$ to 2 by instantiating (for odd i)

- S with the set of tuples of words of the form $\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{i-2} \end{pmatrix}$,
- E with the set of finite words over 2^{AP} ,
- R with the relation $\left\{ \left(\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{i-2} \end{pmatrix}, w_{i-1} \right) \mid |w_{i-1}| = |w_0| \right\}$, and
- \equiv with \equiv_{i+1}^{rf} .

It follows that \equiv' is the relation \equiv_i^{rf} for odd i .

For even i , slightly more work is necessary: We first fix a pair p, q of states of \mathcal{C}_i and then instantiate

- S with the set of tuples of words of the form $\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{i-2} \end{pmatrix}$,
 - E with the set of finite words over 2^{AP} ,
 - R with the relation $\left\{ \left(\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{i-2} \end{pmatrix}, w_{i-1} \right) \mid w_{i-1} \text{ is a } (p, q)\text{-completion of } \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{i-2} \end{pmatrix} \right\}$,
- and
- \equiv with \equiv_{i+1}^{rf} .

The intersection of \equiv_i and the equivalence relations \equiv' obtained by letting p and q range over all pairs of states yields the relation \equiv_i^{rf} for even i . It has finite index, as the intersection of finitely many equivalence relations, all with a finite index, is another equivalence relation of finite index. \square

Let ℓ be minimal such that each word w_0 with $|w_0| \geq \ell$ is in an infinite \equiv_2^{rf} equivalence class. This is well-defined, as \equiv_2^{rf} has finite index, which implies that there are only finitely many words in finite equivalence classes. A block is a word in $(2^{AP})^\ell$.

9.2 The Abstract Game

Now, we have all definitions at hand to define an incomplete information multi-player game $\mathcal{G}(\mathfrak{T}, \varphi)$ that captures the existence of computable Skolem functions. To build some intuition, we first describe the game abstractly and then define it formally as a multi-player graph game in Subsection 9.4.

The game $\mathcal{G}(\mathfrak{T}, \varphi)$ is played between Player U who picks traces for the universally quantified variables (by picking blocks) and a coalition of variable players $\{1, 3, \dots, k-1\}$, who pick traces for the existentially quantified variables (Player i for π_i), also by picking blocks.

As in the intuition given above for the case of a formula of the form $\forall \pi \exists \pi' . \psi$, the rules of the game $\mathcal{G}(\mathfrak{T}, \varphi)$ need to account for the delay inherent to the definition of computable functions. In the $\forall \exists$ setting, this is covered by the fact that the player in charge of π is one block ahead of the player in charge of π' . With more quantifier alternations, we generalize this as follows for $\varphi = \forall \pi_0 \exists \pi_1 \dots \forall \pi_{k-2} \exists \pi_{k-1} . \psi$:

- The player in charge of π_{k-2} is one block ahead of the player in charge of π_{k-1} .
- The player in charge of π_{k-3} must not be ahead of the player in charge of π_{k-2} , but may also not be behind.
- The player in charge of π_{k-4} must be one block ahead of the player in charge of π_{k-3} . This implies that the player in charge of π_{k-4} must be two blocks ahead of the player in charge of π_{k-1} .
- And so on.

So, the player in charge of π_{k-1} picks one block in round 0, the player in charge of π_{k-2} picks two blocks in round 0 (to be one block ahead), the player in charge of π_{k-3} picks two blocks in round 0, the player in charge of π_{k-4} picks three blocks in round 0 and so on. In general, we define $\Delta_i = \frac{k-(i-1)}{2}$ for (odd) $i \in \{1, 3, \dots, k-1\}$ and $\Delta_i = \Delta_{i+1} + 1$ for (even) $i \in \{0, 2, \dots, k-2\}$, e.g., we have $\Delta_{k-1} = 1$, $\Delta_{k-2} = 2$, $\Delta_{k-3} = 2$, and $\Delta_{k-4} = 3$ capturing the “delay” described above.

Now, we split each round $r = 0, 1, 2, \dots$ into subrounds (r, i) for $i = 0, 1, \dots, k-1$.

- In subround $(0, i)$ of round 0 for even i , Player U picks Δ_i blocks $t_{i-1}^0, t_{i-1}^1, \dots, t_{i-1}^{\Delta_i-1}$.
- In subround $(0, i)$ of round 0 for odd i , Player i picks Δ_i blocks $t_i^0, t_i^1, \dots, t_i^{\Delta_i-1}$.
- In subround (r, i) of round $r > 0$ for even i , Player U picks a block $t_{i-1}^{\Delta_i+r}$.
- In subround (r, i) of round $r > 0$ for odd i , Player i picks a block $t_i^{\Delta_i-1+r}$.

Figure 3 illustrates the evolution of a play of the game and illustrates the number of blocks picked in round 0 as well as the resulting “delay” between the selection of blocks for the different variables.

During a play of $\mathcal{G}(\mathfrak{T}, \varphi)$ the players build traces t_0, t_1, \dots, t_{k-1} defined as $t_i = t_i^0 t_i^1 t_i^2 \dots$. We call $(t_0, t_1, \dots, t_{k-1})$ the outcome of the play. The coalition of variable players wins the play if $t_i \notin \text{Tr}(\mathfrak{T})$ for some even i or if $\text{mrg}(t_0, t_1, \dots, t_{k-1}) \in L(\mathcal{P}_k^\mathfrak{T})$, i.e., the variable assignment mapping each π_i to t_i satisfies ψ and each t_i is in $\text{Tr}(\mathfrak{T})$.

As already alluded to above, the game described above is a game of imperfect information, which captures the fact that the Skolem function for an existentially quantified π_i depends only on the universally quantified variables π_j with $j \in \{0, 2, \dots, i-1\}$. Intuitively, we capture this by giving Player i access to all blocks picked in subrounds (r, j) with $j \in \{0, 2, \dots, i-1\}$, but hiding all other picks made by the players in subrounds (r, j) with $j \in \{1, 3, \dots, i-2, i, i+1, i+2, \dots, k-1\}$. Note that Player i not having access to their own moves is not a restriction, as they can always be reconstructed, if necessary.

Formally, a strategy for Player i is a function σ_i mapping sequences of the form

$$\begin{pmatrix} t_0^0 \\ t_2^0 \\ \vdots \\ t_{i-1}^0 \end{pmatrix} \begin{pmatrix} t_1^0 \\ t_3^0 \\ \vdots \\ t_{i-1}^1 \end{pmatrix} \dots \begin{pmatrix} t_0^{\Delta_i+r} \\ t_2^{\Delta_i+r} \\ \vdots \\ t_{i-1}^{\Delta_i+r} \end{pmatrix} \begin{pmatrix} t_0^{\Delta_i+r+1} \\ t_2^{\Delta_i+r+1} \\ \vdots \\ t_{i-3}^{\Delta_i+r+1} \end{pmatrix} \begin{pmatrix} t_0^{\Delta_i+r+2} \\ t_2^{\Delta_i+r+1} \\ \vdots \\ t_{i-5}^{\Delta_i+r+2} \end{pmatrix} \dots \begin{pmatrix} t_0^{\Delta_i+r+\frac{i}{2}-1} \\ t_2^{\Delta_i+r+\frac{i}{2}-1} \end{pmatrix} \begin{pmatrix} t_0^{\Delta_i+r+\frac{i}{2}} \end{pmatrix} \quad (1)$$

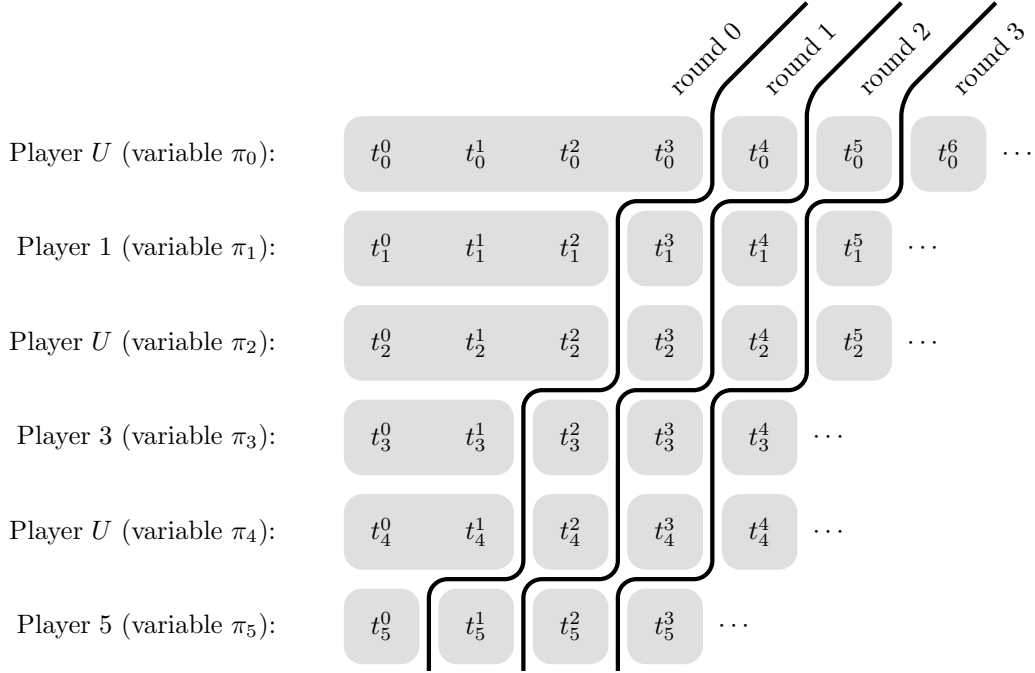


Figure 3: The evolution of a play of $\mathcal{G}(\mathfrak{T}, \varphi)$ for a sentence φ with six variables. Each gray shape is a subround, consisting of a move of Player U or a move of one variable player. We have $\Delta_5 = 1$, $\Delta_4 = \Delta_3 = 2$, $\Delta_2 = \Delta_1 = 3$, and $\Delta_0 = 4$, which corresponds to the number of blocks picked by the player in charge of variable π_i in round 0.

for $r \geq 0$ to a block (or a sequence of Δ_i blocks for $r = 0$). The vectors getting shorter at the end is a manifestation the fact that the players in charge of variables π_j with smaller j are ahead of the players in charge of variables with larger j' (see Figure 3).

A (finite or infinite) play is consistent with σ_i , if the pick of Player i in each round r is the one prescribed by σ_i . A collection $(\sigma_i)_{i \in \{1,3,\dots,k-1\}}$ of strategies, one for each variable player, is winning, if every play that is consistent with all σ_i is won by the variable players. We say that a strategy σ_i is finite-state, if it is implemented by a transducer that reads inputs as in Equation (1) (over some suitable finite alphabet) and produces an output block (or a sequence of Δ_i blocks in round 0).

The following lemma shows that the existence of a winning collection of strategies characterizes the existence of computable Skolem functions. Note that there is a slight mismatch, as the first implication requires the strategies to be finite-state, while the second implication only yields arbitrary strategies. This gap will be closed later.

- Lemma 2.** *1. If the coalition of variable players has a winning collection of finite-state strategies then $\mathfrak{T} \models \varphi$ has computable Skolem functions.*
- 2. If $\mathfrak{T} \models \varphi$ has computable Skolem functions, then the coalition of variable players has a winning collection of strategies.*

Proof. We begin by showing Item 1. So, let $(\sigma_i)_{i \in \{1,3,\dots,k-1\}}$ be a winning collection of finite-state strategies for the variable players. We construct computable Skolem functions $(f_i)_{i \in \{1,3,\dots,k-1\}}$ witnessing $\mathfrak{T} \models \varphi$. So, fix some $i \in \{1,3,\dots,k-1\}$.

The machine \mathcal{M}_i computing f_i works in iterations $n = 0, 1, 2, \dots$ coinciding with the rounds of $\mathcal{G}(\mathfrak{T}, \varphi)$. Its input is $\text{mrg}(t_0, t_2, \dots, t_{i-1})$ (encoding $\frac{i}{2}$ input traces as a single infinite word on the input tape), where we split each t_j into blocks $t_j^0 t_j^1 t_j^2 \dots$. Recall that the block length ℓ is a constant, i.e., \mathcal{M}_i can read its input blockwise.

Now, in iteration 0, \mathcal{M}_i reads the first $\Delta_i + \frac{i}{2}$ blocks of the input, which yields $\Delta_i + \frac{i}{2}$ blocks of each t_j . These blocks can be used to simulate the moves of Player U in subrounds $(0, j)$ for even $j < i$. Note that this does not require all blocks of the t_j for $j > 0$. These have to be stored in the working tape for later use, as the reading tape is one-way. The simulated moves by Player U can be feed into the finite-state implementation of σ_i , yielding blocks $t_i^0, t_i^1, \dots, t_i^{\Delta_i-1}$ as output. The word $t_i^0 t_i^1 \dots t_i^{\Delta_i-1}$ is then written to the output tape of \mathcal{M}_i , which completes iteration 0.

In general, assume \mathcal{M}_i has completed iteration $n - 1$ and now starts iteration $n > 0$. This iteration begins with \mathcal{M}_i reading another block of the input, which yields another block of each t_j . The new block of t_0 , and the oldest stored block for each t_j with $j > 0$ can be used to continue the simulated play (restricted to moves by Player U in subrounds for the variables π_j for $j \in \{0, 2, \dots, i - 1\}$) by feeding them into the finite-state implementation of σ_i , yielding a block t as output. This block is then appended on the output tape. The unused new blocks of t_j with $j > 0$ are again stored on the working tape. This ends iteration n .

To simulate the play, \mathcal{M}_i can just store the whole play prefix on the working tape. To process the play prefix by the finite-state implementation of σ_i , \mathcal{M}_i can just store the whole run prefix on the working tape, although a more economical would be possible.

Now, we show that the functions f_i computed by the \mathcal{M}_i constructed above are indeed Skolem functions witnessing $\mathfrak{T} \models \varphi$. To this end, let Π with $\text{dom}(\Pi) \supseteq \{\pi_0, \pi_1, \dots, \pi_{k-1}\}$ be a variable assignment that is consistent with the f_i , i.e., each $\Pi(\pi_i)$ with even i is in $\text{Tr}(\mathfrak{T})$ and each $\Pi(\pi_i)$ with odd i is equal to $f_i(\Pi(\pi_0), \Pi(\pi_2), \dots, \Pi(\pi_{i-1}))$. We need to show that each $\Pi(\pi_i)$ for odd i is in $\text{Tr}(\mathfrak{T})$ (i.e., the functions f_i are well-defined) and that $\Pi \models \psi$, i.e., $\text{mrg}(\Pi(\pi_0), \Pi(\pi_1), \dots, \Pi(\pi_{k-1})) \in L(\mathcal{P}_k^{\mathfrak{T}})$.

By construction, $(\Pi(\pi_0), \Pi(\pi_1), \dots, \Pi(\pi_{k-1}))$ is the outcome of a play of $\mathcal{G}(\mathfrak{T}, \varphi)$ that is consistent with the σ_i and therefore winning for the variable players. As each $\Pi(\pi_i)$ with even i is in $\text{Tr}(\mathfrak{T})$, we conclude $\text{mrg}(\Pi(\pi_0), \Pi(\pi_1), \dots, \Pi(\pi_{k-1})) \in L(\mathcal{P}_k^{\mathfrak{T}})$, as required. Note that this does also imply $\Pi(\pi_i)$ for odd i is in $\text{Tr}(\mathfrak{T})$, as $L(\mathcal{P}_k^{\mathfrak{T}})$ only contains tuples of traces from $\text{Tr}(\mathfrak{T})$.

Now, let us consider Item 2. Let $f_i: (\text{Tr}(\mathfrak{T}))^{\frac{i+1}{2}} \rightarrow \text{Tr}(\mathfrak{T})$ for $i \in \{1, 3, \dots, k - 1\}$ be computable Skolem functions witnessing $\mathfrak{T} \models \varphi$, say each f_i is implemented by a Turing machine \mathcal{M}_i . By Remark 4, each \mathcal{M}_i can be assumed to have bounded delay: there is a d_i such that to compute the first n letters of the output

only $n + d_i$ letters of the input are read. Note that we can run such a Turing machine \mathcal{M}_i with delay d_i on a finite input w of length $n + d_i$ and obtain the first n letters of the output $f_i(w')$ of every infinite w' that starts with the prefix w . We will apply this fact to simulate the \mathcal{M}_i on-the-fly on longer and longer prefixes.

Also note that our definition of a function f being computed by a Turing machine \mathcal{M} only requires it to compute the output $f(w)$ for all $w \in \text{dom}(f)$, but it can produce arbitrary (even finite) outputs for $w \notin \text{dom}(f)$. To simplify our construction, we assume here that each \mathcal{M}_i produces an infinite output for every input, even if it is not in the domain of f_i . This can be done w.l.o.g., as the \mathcal{M}_i have bounded delay d_i : as soon as \mathcal{M}_i wants to access input letter $n + d_i + 1$ without having produced $n + 1$ output letters so far (this can be detected, as d_i is a constant), the run does not have delay d_i , which implies that the input cannot be in $\text{dom}(f_i)$. Hence, a designated state can be entered, which produces an arbitrary infinite output while ignoring the remaining input. The resulting machine still has delay d_i , but a complete domain.

Let $d_i \in \mathbb{N}$ be minimal such that each \mathcal{M}_i has delay at most d . We inductively define a winning collection of strategies for the variable players.

Round 0.

Subrounds (0, 0) and (0, 1). Assume Player U picks $t_0^0, t_0^1, \dots, t_0^{\Delta_0-1}$ in subround (0, 0) to start a play. We fix $\tilde{t}_0^0 = t_0^0$ and fix \tilde{t}_0^n for $n \in \{1, 2, \dots, \Delta_0 - 1\}$ such that $\tilde{t}_0^n \equiv_2^{\text{rf}} t_0^n$ and $|\tilde{t}_0^0 \tilde{t}_0^1 \dots \tilde{t}_0^n| \geq |\tilde{t}_0^0 \tilde{t}_0^1 \dots \tilde{t}_0^{n-1}| + d$ for all such n . This is always possible, as the \equiv_2^{rf} equivalence class of each t_0^n is infinite and therefore contains arbitrarily long words.

Let $\tilde{t}_1^0 \tilde{t}_1^1 \dots \tilde{t}_1^{\Delta_0-2}$ be the output of \mathcal{M}_1 when given the partial input $\tilde{t}_0^0 \tilde{t}_0^1 \dots \tilde{t}_0^{\Delta_0-1}$ such that $|\tilde{t}_1^n| = |\tilde{t}_0^n|$ for all n . This is well-defined by the choice of the length of the \tilde{t}_0^n and the fact that \mathcal{M}_1 has delay d . Note that \mathcal{M}_1 might produce even more output on that input. Any such additional output is ignored in this subround.

Now, let (q_2^n, c_2^n) be the unique state of \mathcal{C}_2 that is reached by processing $\text{mrg}(\tilde{t}_0^n, \tilde{t}_1^n)$ from $(q_2^{n-1}, \Omega_2(q_2^{n-1}))$, where we use $q_2^{-1} = q_{I,2}$ to start. Again, this is well-defined as \mathcal{C}_2 is deterministic. Each \tilde{t}_1^n is a $((q_2^{n-1}, \Omega_2(q_2^{n-1})), (q_2^n, c_2^n))$ -completion of \tilde{t}_0^n .

As we have $t_0^n \equiv_2^{\text{rf}} \tilde{t}_0^n$ for all such n , there also exists a $((q_2^{n-1}, \Omega_2(q_2^{n-1})), (q_2^n, c_2^n))$ -completion t_1^n of t_0^n for all $n \in \{0, 1, \dots, \Delta_0 - 2\}$ such that $\begin{pmatrix} t_0^n \\ t_1^n \end{pmatrix} \equiv_3^{\text{rf}} \begin{pmatrix} \tilde{t}_0^n \\ \tilde{t}_1^n \end{pmatrix}$. We define σ_1 such that it picks $t_0^0, t_1^0, \dots, t_1^{\Delta_0-2}$ in subround (0, 1). As $\Delta_0 - 2 = \Delta_1 + 1$, these are Δ_1 many blocks, as required by the definition of $\mathcal{G}(\mathfrak{T}, \varphi)$.

Subrounds (0, 2) and (0, 3). Now, assume Player U picks $t_2^0, t_2^1, \dots, t_2^{\Delta_2-1}$ in subround (0, 2). We fix $\tilde{t}_2^0 = t_2^0$ and then fix \tilde{t}_2^n for $n \in \{1, 2, \dots, \Delta_2 - 1\}$ such that $\begin{pmatrix} \tilde{t}_0^n \\ \tilde{t}_1^n \\ \tilde{t}_2^n \end{pmatrix} \equiv_4^{\text{rf}} \begin{pmatrix} t_0^n \\ t_1^n \\ t_2^n \end{pmatrix}$. This is possible, as

we have $\begin{pmatrix} t_0^n \\ t_1^n \end{pmatrix} \equiv_3^{\text{rf}} \begin{pmatrix} \tilde{t}_0^n \\ \tilde{t}_1^n \end{pmatrix}$ for all such n . Let $\tilde{t}_3^0 \tilde{t}_3^1 \dots \tilde{t}_3^{\Delta_2-2}$ be the output of \mathcal{M}_3 when given the partial input $\text{mrg}(\tilde{t}_0^0 \tilde{t}_0^1 \dots \tilde{t}_0^{\Delta_2-1}, \tilde{t}_2^0 \tilde{t}_2^1 \dots \tilde{t}_2^{\Delta_2-1})$ such that $|\tilde{t}_3^n| = |\tilde{t}_0^n|$ for all n (again, this is well-defined due to the choice of the length of the \tilde{t}_0^n and \mathcal{M}_3 having delay d , and might require to ignore some output).

Let (q_4^n, c_4^n) be the unique state of \mathcal{C}_4 that is reached by processing $\text{mrg}(\tilde{t}_0^n, \tilde{t}_1^n, \tilde{t}_2^n, \tilde{t}_3^n)$ from $(q_4^{n-1}, \Omega_4(q_4^{n-1}))$, where we use $q_4^{-1} = q_{I,4}$ to start. Each \tilde{t}_3^n is a $((q_4^{n-1}, \Omega_4(q_4^{n-1})), (q_4^n, c_4^n))$ -completion of $\begin{pmatrix} \tilde{t}_0^n \\ \tilde{t}_1^n \\ \tilde{t}_2^n \end{pmatrix}$.

As we have $\begin{pmatrix} \tilde{t}_0^n \\ \tilde{t}_1^n \\ \tilde{t}_2^n \end{pmatrix} \equiv_4^{\text{rf}} \begin{pmatrix} t_0^n \\ t_1^n \\ t_2^n \end{pmatrix}$ for all such n , there also exists a $((q_4^{n-1}, \Omega_4(q_4^{n-1})), (q_4^n, c_4^n))$ -completion

t_3^n of $\begin{pmatrix} t_0^n \\ t_1^n \\ t_2^n \end{pmatrix}$ for all $n \in \{0, 1, \dots, \Delta_2 - 1\}$ such that $\begin{pmatrix} t_0^n \\ t_1^n \\ \vdots \\ t_3^n \end{pmatrix} \equiv_5^{\text{rf}} \begin{pmatrix} \tilde{t}_0^n \\ \tilde{t}_1^n \\ \vdots \\ \tilde{t}_3^n \end{pmatrix}$. We define σ_3 such that it picks

$t_3^0, t_3^1, \dots, t_3^{\Delta_2-2}$ in subround $(0, 3)$. As $\Delta_2 - 2 = \Delta_3 - 1$, these are Δ_3 many blocks, as required by the definition of $\mathcal{G}(\mathfrak{T}, \varphi)$.

Subrounds $(0, i-1)$ and $(0, i)$ for odd $i \in \{5, 7, \dots, k-1\}$. Assume Player U picks $t_{i-1}^0, t_{i-1}^1, \dots, t_{i-1}^{\Delta_{i-1}-1}$ in subround $(0, i-1)$. As before, we fix $\tilde{t}_{i-1}^0 = t_{i-1}^0$ and then fix \tilde{t}_{i-1}^n for $n \in \{1, 2, \dots, \Delta_{i-1} - 1\}$ such that $\begin{pmatrix} \tilde{t}_0^n \\ \tilde{t}_1^n \\ \vdots \\ \tilde{t}_{i-1}^n \end{pmatrix} \equiv_{i+1}^{\text{rf}} \begin{pmatrix} t_0^n \\ t_1^n \\ \vdots \\ t_{i-1}^n \end{pmatrix}$. This is possible, as $\begin{pmatrix} t_0^n \\ t_1^n \\ \vdots \\ t_{i-2}^n \end{pmatrix} \equiv_i^{\text{rf}} \begin{pmatrix} \tilde{t}_0^n \\ \tilde{t}_1^n \\ \vdots \\ \tilde{t}_{i-2}^n \end{pmatrix}$ for all such n is an invariant of our construction.

Let $\tilde{t}_i^0 \tilde{t}_i^1 \dots \tilde{t}_i^{\Delta_{i-1}-2}$ be the output of \mathcal{M}_i when given the partial input

$$\text{mrg}(\tilde{t}_0^0 \tilde{t}_1^1 \dots \tilde{t}_0^{\Delta_{i-1}-1}, \tilde{t}_2^0 \tilde{t}_2^1 \dots \tilde{t}_2^{\Delta_{i-1}-1}, \dots, \tilde{t}_{i-1}^0 \tilde{t}_{i-1}^1 \dots \tilde{t}_{i-1}^{\Delta_{i-1}-1})$$

such that $|\tilde{t}_i^n| = |\tilde{t}_0^n|$ for all n . As in the previous cases, this is well-defined.

Let (q_i^n, c_i^n) be the unique state of \mathcal{C}_i that is reached by processing $\text{mrg}(\tilde{t}_0^n, \tilde{t}_1^n, \dots, \tilde{t}_i^n)$ from $(q_i^{n-1}, \Omega_i(q_i^{n-1}))$, where we use $q_i^{-1} = q_{I,i}$ to start. Each \tilde{t}_i^n is a $((q_i^{n-1}, \Omega_i(q_i^{n-1})), (q_i^n, c_i^n))$ -completion of $\begin{pmatrix} \tilde{t}_0^n \\ \tilde{t}_1^n \\ \vdots \\ \tilde{t}_{i-1}^n \end{pmatrix}$. As we have

$$\begin{pmatrix} \tilde{t}_0^n \\ \tilde{t}_1^n \\ \vdots \\ \tilde{t}_{i-1}^n \end{pmatrix} \equiv_{i+1}^{\text{rf}} \begin{pmatrix} t_0^n \\ t_1^n \\ \vdots \\ t_{i-1}^n \end{pmatrix} \text{ for all such } n, \text{ there also exists a } ((q_i^{n-1}, \Omega_i(q_i^{n-1})), (q_i^n, c_i^n))\text{-completion } t_i^n \text{ of } \begin{pmatrix} t_0^n \\ t_1^n \\ \vdots \\ t_{i-1}^n \end{pmatrix}$$

for all $n \in \{0, 1, \dots, \Delta_{i-1} - 1\}$ such that $\begin{pmatrix} t_0^n \\ t_1^n \\ \vdots \\ t_i^n \end{pmatrix} \equiv_{i+2}^{\text{rf}} \begin{pmatrix} \tilde{t}_0^n \\ \tilde{t}_1^n \\ \vdots \\ \tilde{t}_i^n \end{pmatrix}$ for all n , satisfying the invariant again. We

define σ_i such that it picks $t_i^0, t_i^1, \dots, t_i^{\Delta_{i-1}-2}$ in subround $(0, i)$. As $\Delta_{i-1} - 2 = \Delta_i - 1$, these are Δ_i many blocks, as required by the definition of $\mathcal{G}(\mathfrak{T}, \varphi)$.

Round $r > 0$. Now, we consider a round $r > 0$, assuming the σ_i are already defined for all earlier rounds. The construction is very similar to the one for round 0, but simpler as each player (also Player U !) only picks a single block in each subround (r, i) of round r .

Subrounds $(r, 0)$ and $(r, 1)$. Assume Player U picks $t_0^{\Delta_0-1+r}$ in subround $(r, 0)$. We fix $\tilde{t}_0^{\Delta_0-1+r}$ such that $t_0^{\Delta_0-1+r} \equiv_2^{\text{rf}} \tilde{t}_0^{\Delta_0-1+r}$ and $|\tilde{t}_0^0 \tilde{t}_0^1 \dots \tilde{t}_0^{\Delta_0-1+r}| \geq |\tilde{t}_0^0 \tilde{t}_0^1 \dots \tilde{t}_0^{\Delta_0+r-2}| + d$. This is possible, as the \equiv_2^{rf} equivalence class of $\tilde{t}_0^{\Delta_0-1+r}$ is infinite and therefore contains arbitrarily long words.

We run \mathcal{M}_1 on $\tilde{t}_0^0 \tilde{t}_0^1 \dots \tilde{t}_0^{\Delta_0-1+r}$ and obtain another block $\tilde{t}_1^{\Delta_0+r-2}$. Then, we define $(q_2^{\Delta_0+r-2}, c_2^{\Delta_0+r-2})$ as the unique state of \mathcal{C}_2 reached from $(q_2^{\Delta_0-1+r-2}, \Omega_2(q_2^{\Delta_0-1+r-2}))$ (which was defined in subround $(r-1, 0)$) when processing $\text{mrg}(\tilde{t}_0^{\Delta_0+r-2}, \tilde{t}_1^{\Delta_0+r-2})$, i.e., $\tilde{t}_1^{\Delta_0+r-2}$ is a $((q_2^{\Delta_0-1+r-2}, c_2^{\Delta_0-1+r-2}), (q_2^{\Delta_0+r-2}, \Omega_2(q_2^{\Delta_0+r-2})))$ -completion of $\tilde{t}_0^{\Delta_0+r-2}$.

There is also a $((q_2^{\Delta_0-1+r-2}, c_2^{\Delta_0-1+r-2}), (q_2^{\Delta_0+r-2}, \Omega_2(q_2^{\Delta_0+r-2})))$ -completion $t_1^{\Delta_0+r-2}$ of $\tilde{t}_0^{\Delta_0+r-2}$ such that $\begin{pmatrix} t_0^{\Delta_0+r-2} \\ t_1^{\Delta_0+r-2} \end{pmatrix} \equiv_3^{\text{rf}} \begin{pmatrix} \tilde{t}_0^{\Delta_0+r-2} \\ \tilde{t}_1^{\Delta_0+r-2} \end{pmatrix}$, as we have $t_0^{\Delta_0+r-2} \equiv_2^{\text{rf}} \tilde{t}_0^{\Delta_0+r-2}$. We define σ_1 such that it picks the block $t_1^{\Delta_0+r-2}$ in subround $(r, 1)$ (note that $\Delta_0 + r - 2 = \Delta_1 - 1 + r$).

Subrounds $(r, i-1)$ and (r, i) for $i \in \{3, 5, \dots, k-1\}$. Now, assume Player U picks $t_{i-1}^{\Delta_{i-1}-1+r}$ in subround $(r, i-1)$. We fix $\tilde{t}_{i-1}^{\Delta_{i-1}-1+r}$ such that
$$\begin{pmatrix} \tilde{t}_0^{\Delta_{i-1}-1+r} \\ \tilde{t}_1^{\Delta_{i-1}-1+r} \\ \vdots \\ \tilde{t}_{i-1}^{\Delta_{i-1}-1+r} \end{pmatrix} \equiv_{i+1}^{\text{rf}} \begin{pmatrix} t_0^{\Delta_{i-1}-1+r} \\ t_1^{\Delta_{i-1}-1+r} \\ \vdots \\ t_{i-1}^{\Delta_{i-1}-1+r} \end{pmatrix}. \quad \text{This is possible, as}$$

$$\begin{pmatrix} \tilde{t}_0^{\Delta_{i-1}-1+r} \\ \tilde{t}_1^{\Delta_{i-1}-1+r} \\ \vdots \\ \tilde{t}_{i-2}^{\Delta_{i-1}-1+r} \end{pmatrix} \equiv_i^{\text{rf}} \begin{pmatrix} t_0^{\Delta_{i-1}-1+r} \\ t_1^{\Delta_{i-1}-1+r} \\ \vdots \\ t_{i-2}^{\Delta_{i-1}-1+r} \end{pmatrix}$$
 is an invariant of our construction.

We run \mathcal{M}_i on

$$\text{mrg}(\tilde{t}_0^0 \tilde{t}_1^1 \dots \tilde{t}_0^{\Delta_{i-1}-1+r}, \tilde{t}_2^0 \tilde{t}_1^1 \dots \tilde{t}_2^{\Delta_{i-1}-1+r}, \dots, \tilde{t}_{i-1}^0 \tilde{t}_{i-1}^1 \dots \tilde{t}_{i-1}^{\Delta_{i-1}-1+r}),$$

yielding another block $\tilde{t}_i^{\Delta_{i-1}+r-2}$. Let $(q_i^{\Delta_{i-1}+r-2}, c_i^{\Delta_{i-1}+r-2})$ be the unique state of \mathcal{C}_i that is reached by processing

$$\text{mrg}(\tilde{t}_0^{\Delta_{i-1}+r-2}, \tilde{t}_1^{\Delta_{i-1}+r-2}, \dots, \tilde{t}_i^{\Delta_{i-1}+r-2})$$

from $(q_i^{\Delta_{i-1}-1+r-2}, \Omega_i(q_i^{\Delta_{i-1}-1+r-2}))$ (defined in subround $(r-1, i-1)$). Hence, $\tilde{t}_i^{\Delta_{i-1}+r-2}$ is a

$$((q_i^{\Delta_{i-1}-1+r-2}, \Omega_i(q_i^{\Delta_{i-1}-1+r-2})), (q_i^{\Delta_{i-1}+r-2}, c_i^{\Delta_{i-1}+r-2}))\text{-completion of } \begin{pmatrix} \tilde{t}_0^{\Delta_{i-1}+r-2} \\ \tilde{t}_1^{\Delta_{i-1}+r-2} \\ \vdots \\ \tilde{t}_{i-1}^{\Delta_{i-1}+r-2} \end{pmatrix}. \quad \text{As } \begin{pmatrix} \tilde{t}_0^{\Delta_{i-1}+r-2} \\ \tilde{t}_1^{\Delta_{i-1}+r-2} \\ \vdots \\ \tilde{t}_{i-1}^{\Delta_{i-1}+r-2} \end{pmatrix} \equiv_{i+1}^{\text{rf}}$$

$$\begin{pmatrix} t_0^{\Delta_{i-1}+r-2} \\ t_1^{\Delta_{i-1}+r-2} \\ \vdots \\ t_{i-1}^{\Delta_{i-1}+r-2} \end{pmatrix}, \text{ there also exists a}$$

$((q_i^{\Delta_{i-1}-1+r-2}, \Omega_i(q_i^{\Delta_{i-1}-1+r-2})), (q_i^{\Delta_{i-1}+r-2}, c_i^{\Delta_{i-1}+r-2}))\text{-completion } t_i^{\Delta_{i-1}+r-2}$ of
$$\begin{pmatrix} t_0^{\Delta_{i-1}+r-2} \\ t_1^{\Delta_{i-1}+r-2} \\ \vdots \\ t_{i-1}^{\Delta_{i-1}+r-2} \end{pmatrix} \text{ such that } \begin{pmatrix} t_0^{\Delta_{i-1}+r-2} \\ t_1^{\Delta_{i-1}+r-2} \\ \vdots \\ t_{i-1}^{\Delta_{i-1}+r-2} \end{pmatrix} \equiv_{i+2}^{\text{rf}} \begin{pmatrix} \tilde{t}_0^{\Delta_{i-1}+r-2} \\ \tilde{t}_1^{\Delta_{i-1}+r-2} \\ \vdots \\ \tilde{t}_{i-1}^{\Delta_{i-1}+r-2} \end{pmatrix}.$$
 We define σ_i such that it picks $t_i^{\Delta_{i-1}+r-2}$ in subround (r, i) (note that $\Delta_{i-1} + r - 2 = \Delta_i - 1 + r$).

This completes the definition of the σ_i . Note that each σ_i does indeed only depend on the blocks picked in subrounds (r, j) with $j \in \{0, 2, \dots, i-1\}$, i.e., σ_i is indeed a strategy for Player i in $\mathcal{G}(\mathfrak{T}, \varphi)$.

It remains to show that the σ_i are a winning collection of strategies. To this end, let $(t_0, t_1, \dots, t_{k-1})$ be an outcome of a play that is consistent with the σ_i . If a t_i with even i is not in $\text{Tr}(\mathfrak{T})$, then the variable players win immediately. So, assume each t_i with even i is in $\text{Tr}(\mathfrak{T})$. Let $\tilde{t}_0, \tilde{t}_1, \dots, \tilde{t}_{k-1}$ be the traces constructed during the inductive definition of the σ_i . By applying Remark 7.2 and the fact that \equiv_k^{rf} refines \equiv_k , we obtain that each \tilde{t}_i with even i is in $\text{Tr}(\mathfrak{T})$ as well. Also, the \tilde{t}_i for odd i satisfy $\tilde{t}_i = f_i(\tilde{t}_0, \tilde{t}_2, \dots, \tilde{t}_{i-1})$ by construction, i.e., they are obtained by applying the Skolem functions. Hence, the variable assignment mapping π_i to \tilde{t}_i satisfies ψ , which implies that $\text{mrg}(\tilde{t}_0, \tilde{t}_1, \dots, \tilde{t}_{i-1})$ is in $L(\mathcal{P}_k^{\mathfrak{T}})$. Now, consider the sequence of states $(q_k^{-1}, \Omega_k(q_k^{-1})), (q_k^0, c_k^0), (q_k^1, c_k^1), (q_k^2, c_k^2), \dots$ obtained during the definition of σ_{k-1} . They satisfy $\mathcal{C}_k : (q_k^{n-1}, \Omega_k(q_k^{n-1})) \xrightarrow{\text{mrg}(\tilde{t}_0^n, \tilde{t}_1^n, \dots, \tilde{t}_{k-1}^n)} (q_k^n, c_k^n)$ for all $n \in \mathbb{N}$ and $(q_k^{-1}, \Omega_k(q_k^{-1}))$ is the initial state of \mathcal{C}_k . Hence, Remark 6.2 implies that $c_k^0 c_k^1 c_k^2 \dots$ satisfies the parity condition.

Now, as $\begin{pmatrix} \tilde{t}_0^n \\ \tilde{t}_1^n \\ \vdots \\ \tilde{t}_{k-1}^n \end{pmatrix} \equiv_k \begin{pmatrix} t_0^n \\ t_1^n \\ \vdots \\ t_{k-1}^n \end{pmatrix}$ for all n , we have $\mathcal{C}_k: (q_k^{n-1}, \Omega_k(q_k^{n-1})) \xrightarrow{\text{mrg}(t_0^n, t_1^n, \dots, t_{k-1}^n)} (q_k^n, c_k^n)$ for all n

as well. By applying Remark 6.2 again, we obtain $\text{mrg}(t_0, t_1, \dots, t_{i-1}) \in L(\mathcal{P}_k^\mathcal{T})$, i.e., the variable players win. \square

9.3 Distributed Games

After having shown that the abstract game $\mathcal{G}(\mathcal{T}, \varphi)$ characterizes the existence of computable Skolem functions, we now model $\mathcal{G}(\mathcal{T}, \varphi)$ as a multi-player graph game of imperfect information using the notation of Berwanger et al. [4]. In this subsection, we introduce the necessary definitions before we model the game in Subsection 9.4. The games considered by Berwanger et al. are concurrent games (i.e., the players make their moves simultaneously), while $\mathcal{G}(\mathcal{T}, \varphi)$ is turn-based, i.e., the players make their moves one after the other. Hence, we will also introduce some notation for the special case of turn-based games, which simplifies our modeling.

Game Graphs. Fix some set $N = \{1, \dots, n\}$ of players and a distinguished agent called Nature (which is not in N !). A profile is a list $p = (p_1, \dots, p_n)$ of elements $p_i \in P_i$ for sets P_i that will be clear from context. For each player $i \in N$ we fix a finite set A^i of actions and a finite set B^i of observations. A game graph $G = (V, E, v_I, (\beta^i)_{i \in N})$ consists of a finite set V of positions, an edge relation $E \subseteq V \times A \times V$ representing simultaneous moves labeled by action profiles (i.e., $A = A^1 \times \dots \times A^n$), an initial position $v_I \in V$, and a profile $(\beta^i)_{i \in N}$ of observation functions $\beta^i: V \rightarrow B^i$ that label, for each player, the positions with observations. We require that E has no dead ends, i.e., for every $v \in V$ and every $a \in A$ there is a $v' \in V$ with $(v, a, v') \in E$.

A game graph $(V, E, v_I, (\beta^i)_{i \in N})$ yields hierarchical information if there exists a total order \preceq over N such that if $i \preceq j$ then for all $v, v' \in V$, $\beta^i(v) = \beta^i(v')$ implies $\beta^j(v) = \beta^j(v')$, i.e., if Player i cannot distinguish v and v' , then neither can Player j for $i \preceq j$.

Plays. Intuitively, a play starts at position $v_I \in V$ and proceeds in rounds. In a round at position v , each Player i chooses simultaneously and independently an action $a^i \in A^i$, then Nature chooses a successor position v' such that $(v, a, v') \in E$. Now, each player receives the observation $\beta^i(v')$ and the next round is played at position v' . Thus, a play of G is an infinite sequence $v_0 v_1 v_2 \dots$ of vertices such that $v_0 = v_I$ and for all $r \geq 0$ there is an $a_r \in A$ such that $(v_r, a_r, v_{r+1}) \in E$.

A history is a prefix $v_0 v_1 \dots v_r$ of a play. We denote the set of all histories by $\text{Hist}(G)$ and extend $\beta^i: V \rightarrow B^i$ to plays and histories by defining $\beta^i(v_0 v_1 v_2 \dots) = \beta^i(v_1) \beta^i(v_2) \beta^i(v_3) \dots$. Note that the observation of the initial position is discarded for technical reasons [4]. We say two histories h and h' are indistinguishable to Player i , denoted by $h \sim_i h'$, if $\beta^i(h) = \beta^i(h')$.

Strategies. A strategy for Player i is a mapping $s^i: V^* \rightarrow A^i$ that satisfies $s^i(h) = s^i(h')$ for all h, h' with $h \sim_i h'$ (i.e., the action selected by the strategy only depends on the observations of the history). A play $v_0 v_1 v_2 \dots$ is consistent with s^i if for every $r \geq 0$, there is an $a_r = (a^1, \dots, a^n) \in A$ with $(v_r, a_r, v_{r+1}) \in E$ and $a^i = s^i(v_0 v_1 \dots v_r)$. A play is consistent with a strategy profile (s^1, \dots, s^n) if it is consistent with each s^i . The set of possible outcomes of a strategy profile is the set of all plays that are consistent with s .

A distributed game $\mathcal{G} = (G, W)$ consists of a game graph and a winning condition $W \subseteq V^\omega$, where V is the set of positions of G . A play is winning if it is in W and a strategy profile S is winning in \mathcal{G} if all its outcomes are winning.

Finite-state Strategies. Next, we define what it means for a strategy for Player i to be finite-state. So far, we used transducers, i.e., automata with output on transitions to implement strategies in a finitary

manner. From now on, we follow the definitions used by Berwanger et al. [4] and use Moore machines, i.e., finite automata with output on states. However, each Moore machine can be transformed into a transducer by “moving” the output from a state to all its outgoing transitions.

Let A^i and B^i be the actions and observations of Player i . A Moore machine $\mathcal{S} = (M, m_I, \text{upd}, \text{nxt})$ for Player i consists of a finite set M of memory states containing the initial memory state m_I , a memory update function $\text{upd}: M \times B^i \rightarrow M$, and a next-move function $\text{nxt}: M \rightarrow A^i$. We extend upd to words over B^i by defining $\text{upd}(\varepsilon) = m_I$ and $\text{upd}(b_0 b_1 \cdots b_r) = \text{upd}(\text{upd}(b_0 b_1 \cdots b_{r-1}), b_r)$. We say \mathcal{S} implements the strategy mapping $v_0 v_1 \cdots v_r$ to $\text{nxt}(\text{upd}(\beta^i(v_0 v_1 \cdots v_r)))$. A strategy is finite-state, if it is implemented by some Moore machine.

Proposition 4 ([29, 4]).

1. *The following problem is decidable: Given a distributed game with ω -regular winning condition, does it have a winning strategy profile?*
2. *A distributed game with ω -regular winning condition has a winning strategy profile if and only if it has a winning profile of finite-state strategies.*

Turn-based Game Graphs. We say that a game graph $G = (V, E, v_I, (\beta^i)_{i \in N})$ is turn-based, if there is a function $o: V \rightarrow N$ such that if $(v, a, v') \in E$, $(v, a', v'') \in E$, and the action profiles a and a' having the same action for Player $o(v)$ implies $v' = v''$. To simplify our notation, we label the edges leaving v only by actions of Player $o(v)$. Thus, in a turn-based game graph, at every position v Player $o(v)$ determines the possible next moves, and Nature selects one of them. Turn-based distributed games are distributed games whose game graphs are turn-based.

9.4 The Concrete Game

Now, we are finally ready to formalize the abstract game $\mathcal{G}(\mathfrak{T}, \varphi)$ described above as a turn-based distributed game.

We begin by introducing notation for the positions of the game, which intuitively keep track of blocks picked by the players of $\mathcal{G}(\mathfrak{T}, \varphi)$ until they can be processed by an automaton recognizing the winning condition. Due to the delay between the choices by the different players, this requires some notation.

Recall that we have defined Δ_i for even i to be the number of blocks Player U picks in subround $(0, i)$ for variable i and for odd i to be the number of blocks Player i picks in subround $(0, i)$ for variable i . Now, let

$$D = \{(j, x) \mid j \in \{0, 1, \dots, k-1\} \text{ and } x \in \{0, 1, \dots, \Delta_i - 1\}\}.$$

A configuration is a partial function $c: D \rightarrow \mathcal{B}$, where $\mathcal{B} = (2^{\text{AP}})^\ell$ denotes the set of blocks. Let C denote the set of all configurations. The following definitions are visualized in Figure 4.

We only need certain types of configurations c for our construction. We say that c is

- a full configuration if $\text{dom}(c) = D$,
- an initial i -configuration (for $i \in \{0, 1, \dots, k-1\}$) if

$$\text{dom}(c) = \{(j, x) \mid j \in \{0, 1, \dots, i-1\} \text{ and } x \in \{0, 1, \dots, \Delta_j - 1\}\},$$

and

- a looping i -configuration (again for $i \in \{0, 1, \dots, k-1\}$) if

$$\begin{aligned} \text{dom}(c) = \{(j, x) \mid j \in \{0, 1, \dots, i-1\} \text{ and } x \in \{0, 1, \dots, \Delta_j - 1\}\} \cup \\ \{(j, x) \mid j \in \{i, i+1, \dots, k-1\} \text{ and } x \in \{0, 1, \dots, \Delta_j - 2\}\}. \end{aligned}$$

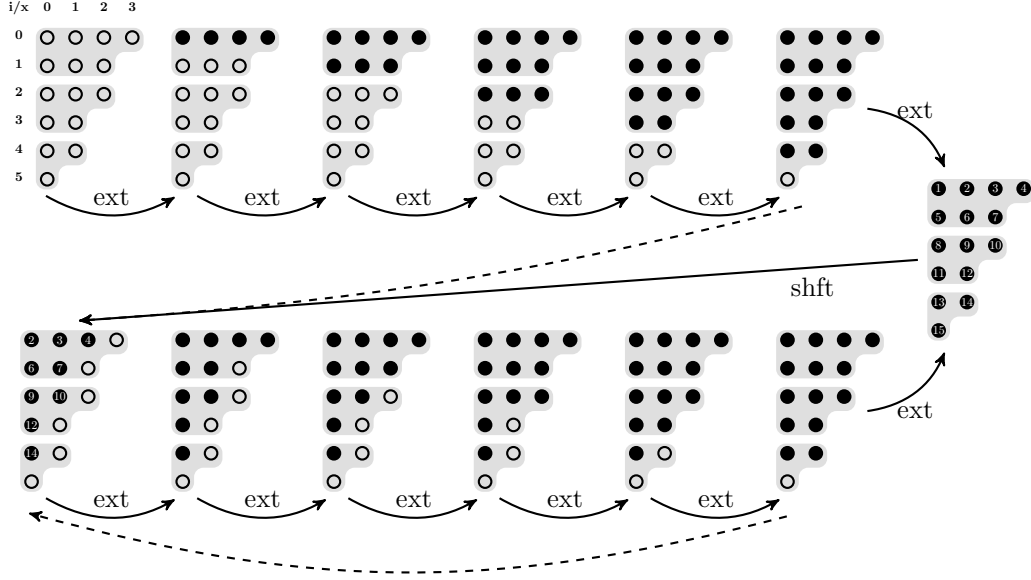


Figure 4: Illustrating configurations for a formula with six variables, where a filled circle denotes an element in the domain, and an unfilled circle an element that is not in the domain. The upper row shows initial i -configurations for $i = 0, 1, 2, \dots, 5$ (from left to right), the lower row shows looping i -configurations for $i = 0, 1, 2, \dots, 5$ (from left to right), and the configuration on the right in between the rows is full. Solid arrows show the effect of extending and shifting configurations. The shifting operation is illustrated using the numbers in the circles. Finally, in a play of $\mathcal{G}(\mathcal{T}, \varphi)$ the configurations stored in positions follow the solid arrows, but take the dashed shortcuts avoiding full configurations.

Note that for $i = k - 1$, both the definition of initial and looping i -configuration coincides, as we have $\Delta_{k-1} = 1$. Hence, in the following, we will just speak of $(k - 1)$ -configurations whenever convenient.

Given an initial i -configuration c and a sequence $\bar{b} = b_0, b_1, \dots, b_{\Delta_i-1}$ of Δ_i blocks, we define $\text{ext}(c, \bar{b})$ to be the configuration c' defined as

$$c'(j, x) = \begin{cases} c(j, x) & \text{if } (j, x) \in \text{dom}(c), \\ b_x & \text{if } j = i, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Furthermore, given a looping i -configuration c and a block b , we define $\text{ext}(c, b)$ to be the configuration c' defined as

$$c'(j, x) = \begin{cases} c(j, x) & \text{if } (j, x) \in \text{dom}(c), \\ b & \text{if } j = i \text{ and } x = \Delta_i - 1, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Finally, given a full configuration c , we define $\text{shft}(c)$ to be the configuration c' defined as

$$c'(j, x) = \begin{cases} c(j, x + 1) & \text{if } x < \Delta_j - 1, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The following remark collects how these operations update initial and looping configurations.

Remark 8.

1. If c is an initial i -configuration for $i < k - 1$, then $\text{ext}(c, \bar{b})$ is an initial $(i + 1)$ -configuration.
2. If c is a $(k - 1)$ -configuration, then $\text{ext}(c, b)$ is a full configuration.
3. If c is a full configuration, then $\text{shft}(c)$ is a looping 0-configuration.
4. If c is a looping i -configuration for $i < k - 1$, then $\text{ext}(c, b)$ is a looping $(i + 1)$ -configuration.

Finally, let $\mathcal{P} = (Q, (2^{\text{AP}})^k, q_I, \delta, \Omega)$ be a deterministic parity automaton accepting the language of words of the form $\text{mrg}(t_0, t_1, \dots, t_{k-1}) \in ((2^{\text{AP}})^\omega)^k$ such that either $t_i \notin \text{Tr}(\mathfrak{T})$ for some even i or if $\text{mrg}(t_0, t_1, \dots, t_{k-1}) \in L(\mathcal{P}_k^\mathfrak{T})$ (i.e., accepting the winning outcomes of plays of $\mathcal{G}(\mathfrak{T}, \varphi)$).

Now, we are able to formally define $\mathcal{G}(\mathfrak{T}, \varphi)$. It will be played by the players in $N = \{1, 3, \dots, k - 1\}$ (ignoring, for the sake of readability, the fact that N is not of the form $\{1, 2, \dots, n\}$ for some n , as required by the definitions in Subsection 9.3). Furthermore, the role of Player U will be played by Nature.

We define the set of positions to contain all tuples (i, c, q) where $i \in \{0, 1, \dots, k - 1\}$, c is an (initial or looping) i -configuration, and q is a state of \mathcal{P} , together with a sink state s_\perp . The initial position is $(0, c_\perp, q_I)$ where c_\perp is the configuration with empty domain (which is the unique initial 0-configuration).

We define the action set for Player i (for odd i) as $\mathcal{B}^{\Delta_i} \cup \mathcal{B}$, where actions in \mathcal{B}^{Δ_i} are intended for round 0 and those in \mathcal{B} are intended for all other rounds. If the wrong action is used, then the sink state will be reached. Next, we define the function o determining which player picks an action to continue a play: we have $o(i, c, q) = i$ for odd i , $o(i, c, q) = 1$ for even i (we will soon explain how Player U moves are simulated even though Player 1 owns the corresponding positions), and $o(s_\perp) = 1$.

The set E of edges is defined as follows (recall that we label edges by actions of a single player, as we define a turn-based game):

- We begin by modelling the moves of Player U . Recall that in a turn-based distributed game, Nature resolves the nondeterminism left after the player who is in charge at that positions has picked an action. Thus, we simulate a move of Player U by giving the position to (say) Player 1. Then, we define the edges such that Player 1's move is irrelevant, but the nondeterminism models the choice of Player U . Formally, for $i \in \{0, 2, \dots, k - 2\}$, an initial i -configuration c , and a state q of \mathcal{P} , we have the edge $((i, c, q), a, (i + 1, \text{ext}(c, \bar{b}), q))$ for every action a of Player 1 and every $\bar{b} \in \mathcal{B}^{\Delta_i}$: No matter which action a Player 1 picks, Nature can for each possible \bar{b} pick a successor that extends c by \bar{b} . This indeed simulates the move of Player U in subround $(0, i)$.
- For $i \in \{1, 3, \dots, k - 3\}$, an initial i -configuration c , and a state q of \mathcal{P} , we have the edge $((i, c, q), \bar{b}, (i + 1, \text{ext}(c, \bar{b}), q))$ for each $\bar{b} \in \mathcal{B}^{\Delta_i}$ (this simulates the move of Player i in subround $(0, i)$) as well as the edge $((i, c, q), b, s_\perp)$ for each $b \in \mathcal{B}$ (Player i may not pick a single block in subround $(0, i)$ if $i < k - 1$). Note that there is no nondeterminism to resolve for Nature, as there is a unique successor position for each action.
- For a $(k - 1)$ -configuration c and state q of \mathcal{P} , we have the edge $((k - 1, c, q), b, (0, \text{shft}(c'), q'))$ for each $b \in \mathcal{B}$ (recall that we have $\Delta_{k-1} = 1$ here), where $c' = \text{ext}(c, b)$ and q' is the state reached by \mathcal{P} when processing $\text{mrg}(c'(0, 0), c'(1, 0), \dots, c'(k - 1, 0))$ from q .
- For $i \in \{0, 2, \dots, k - 2\}$, a looping i -configuration c , and a state q of \mathcal{P} , we have the edge $((i, c, q), a, (i + 1, \text{ext}(c, b), q))$ for every action a of Player 1 and every $b \in \mathcal{B}$: No matter which action a Player 1 picks, Nature can for each possible b pick a successor that extends c by b . This simulates the move of Player U in a subround $(r, i + 1)$ for $r > 0$ using the same mechanism as described above for moves of Player U in initial configurations.
- For $i \in \{1, 3, \dots, k - 3\}$, a looping i -configuration c , and a state q of \mathcal{P} , we have the edge $((i, c, q), b, (i + 1, \text{ext}(c, b), q))$ for each $b \in \mathcal{B}$ (this simulates the move of Player i in subround (r, i) for $r > 0$) as well as the edge $((i, c, q), \bar{b}, s_\perp)$ for each $\bar{b} \in \mathcal{B}^{\Delta_i}$ (Player i may not pick a sequence of blocks in a subround (r, i)). Again, there is no nondeterminism to resolve for Nature, as there is a unique successor position for each action.

- For completeness, we have the edge (s_\perp, a, s_\perp) for every action a of Player 1.

It remains to define the observation functions β^i as $\beta^i(i, c, q) = c \upharpoonright i$ where $c \upharpoonright i$ is the configuration defined as

$$(c \upharpoonright i)(j, x) = \begin{cases} c(j, x) & \text{if } j \in \{0, 2, \dots, i-1\}, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

i.e., Player i can only observe the blocks picked for the variables π_j with even $j < i$. For completeness, we also define the observation of the sink state s_\perp to be \perp , where $\perp \notin C$. Then, the order $1 \succeq 3 \succeq \dots \succeq k-1$ witnesses that the game graph yields hierarchical information.

This completes the definition of the game graph. To complete the definition of the game we define the winning condition as follows: Let $(i_0, c_0, q_0)(i_1, c_1, q_1)(i_2, c_2, q_2) \dots$ be a play and let $(i_0, c_0, q_0)(i_k, c_k, q_k)(i_{2k}, c_{2k}, q_{2k}) \dots$ be the subsequence of all (initial and looping) 0-configurations. Recall that the state q_{rk} in such a position (for $r > 0$) is obtained by processing some $\text{mrg}(b_0, \dots, b_{k-1})$ from $q_{(r-1)k}$, where the b_i are blocks picked by the players. We say $(i_0, c_0, q_0)(i_1, c_1, q_1)(i_2, c_2, q_2) \dots$ is in the winning condition, if $\Omega(q_0)\Omega(q_k)\Omega(q_{2k}) \dots$ satisfies the parity condition, which is an ω -regular winning condition. In particular, no winning play may visit the sink s_\perp .

Remark 9. *The (concrete) distributed game constructed here is a formalization of the abstract game $\mathcal{G}(\mathfrak{T}, \varphi)$ described in Subsection 9.2. In particular, a winning collection of (finite-state) strategies for the coalition of players in the abstract game corresponds to a winning (finite-state) strategy profile in the concrete game and vice versa.*

Hence, whenever convenient below, we do not distinguish between the concrete and the abstract game.

Now, our main theorem (Theorem 3) is a direct consequence of Proposition 4 and Lemma 2. Recall that we need to show that the following problem is decidable: “Given a transition system \mathfrak{T} and a HyperLTL sentence φ with $\mathfrak{T} \models \varphi$, is $\mathfrak{T} \models \varphi$ witnessed by computable Skolem functions?”, i.e., does $\mathfrak{T} \models \varphi$ have an explanation in the cs-paradigm? If the answer is yes, our algorithm computes bounded-delay transducers implementing such Skolem functions.

Proof. Due to Lemma 2 and Proposition 4.2, the following statements are equivalent:

- $\mathfrak{T} \models \varphi$ has computable Skolem functions.
- $\mathcal{G}(\mathfrak{T})$ has a winning strategy profile.
- $\mathcal{G}(\mathfrak{T})$ has a winning profile of finite-state strategies.

The last statement can be decided effectively due to Proposition 4.1. Thus, the existence of computable Skolem functions is decidable.

Now, assume $\mathfrak{T} \models \varphi$ has computable Skolem functions. Due to the equivalence above, (the concrete) $\mathcal{G}(\mathfrak{T}, \varphi)$ has a winning profile $(s^1, s^3, \dots, s^{k-1})$ of finite-state strategies. We show by induction over i how the finite-state strategies s^i can be turned into bounded-delay transducers \mathcal{T}_i computing Skolem functions witnessing $\mathfrak{T} \models \varphi$. The proof follows closely the analogous results shown in Lemma 2.1 for Turing machine computable Skolem functions.

Let us fix some $i \in \{1, 3, \dots, k-1\}$ and let \mathcal{S}_i be a Moore machine for Player i implementing s^i . Recall that \mathcal{S}_i reads observations of Player i (configurations of the form $c \upharpoonright i$ for initial and looping configurations) and returns actions of Player i .

On the other hand, \mathcal{T}_i reads an input $\text{mrg}(t_0, t_2, \dots, t_{i-1}) \in ((2^{\text{AP}})^{\frac{i}{2}})^\omega$ where we split each t_j into blocks $t_j = t_j^0 t_j^1 t_j^2 \dots$.

We construct \mathcal{T}_i so that it works in two phases, an initialization phase and a looping phase, that is repeated ad infinitum. We begin by describing the initialization. It begins by \mathcal{T}_i reading Δ_0 blocks from each t_j . These blocks can now be assembled into a sequence of observations of Player i corresponding to the play prefix of $\mathcal{G}(\mathfrak{T}, \varphi)$ in which Player U picks these blocks. Note that this requires to process each such observation twice, as Player i 's observation does not get updated, when Player j for odd $j < i$ makes

a move. All unused blocks are stored in the state space of \mathcal{T} . Now, \mathcal{T}_i simulates the run of the Moore machine \mathcal{S}_i implementing s^i on this sequence of observations, yielding an action a^i . As s^i is winning, this action is a sequence $\bar{b} = b_0, b_1, \dots, b_{\Delta_i-1}$ of blocks. Then, \mathcal{T}_i outputs $b_0 b_1 \dots b_{\Delta_i-1}$. Now, we process the last observation another $k - i$ times with \mathcal{S}_i , simulating the moves for the remaining variables (which are hidden from Player i which implies the observation is unchanged). This concludes the initialization phase.

The looping phase begins with \mathcal{T}_i reading another block from each t_j . Again, these blocks and the ones stored in the state space can be assembled into observations of Player i corresponding to a continuation of the simulated play prefix in which Player U pick these blocks. Then, the run of the Moore machine \mathcal{S}_i implementing s^i can be continued, yielding an action a^i . This is now a block b , which is output by \mathcal{T}_i . Again, we process the last observation just assembled another $k - i$ times to simulate the moves for the remaining variables, which concludes one looping phase. Note that the delay of \mathcal{T}_i is bounded by $k \cdot \ell$, where ℓ is the block length.

By storing blocks from the input that have been read but not yet used in observations, by discarding blocks no longer needed, and by keeping track of the state the simulated run of \mathcal{S}_i ends in, this behaviour can indeed be implemented using a finite state-space. We leave the tedious, but straightforward, formal definition of \mathcal{T}_i to the reader. \square