PROBABILITY DISTRIBUTIONS COMPUTED BY HARD-ATTENTION TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Most expressivity results for transformers treat them as language recognizers (which accept or reject strings), and not as they are used in practice, as language models (which generate strings autoregressively and probabilistically). Here, we characterize the probability distributions that transformer language models can express. We show that making transformer language recognizers autoregressive can sometimes increase their expressivity, and that making them probabilistic can break equivalences that hold in the non-probabilistic case. Our overall contribution is to tease apart what functions transformers are capable of expressing, in their most common use-case as language models.

1 INTRODUCTION

Most work studying transformer expressivity, that is, what classes of computations transformers can perform, treats them as *language recognizers*, where the input is a string and the output is a binary classification: true if the string is accepted and false otherwise (Strobl et al., 2024). However, the most common practical use of transformers is as *language models* (LMs), which differ in two ways: first, the input is a prefix of a string, and the output is a prediction of the next symbol; second, the prediction is a probability distribution rather than a binary decision. Such probability distributions, when estimated from large text corpora, have enabled a wide range of applications in natural language processing and beyond. An open and fundamental question concerns which probability distributions transformer language models can express. This distributional perspective exposes where previously-proven equivalences retain or lose their validity in the probabilistic setting.

We distinguish, on the one hand, between *unweighted* (or equivalently, *Boolean-weighted*) and *real-weighted* computation, and, on the other hand, between *classifiers*, which map a complete string to a value, and *autoregressors*, which map each prefix to a distribution over the next token. Under this terminology, most theoretical work on transformer expressivity (e.g. Yang et al., 2024; Jerad et al., 2025) focuses on Boolean classifiers, while practical applications use transformers as real-weighted autoregressors. The theoretical expressivity of transformers as real autoregressors has been comparatively underexplored.

In this paper, we answer this question for several variants of transformers. Yang et al. (2024) proved that strictly-masked rightmost UHATs, as language recognizers, recognize the same languages as linear temporal logic (LTL) and counter-free automata. There are two commonly-used weighted analogues of regular languages, those defined by weighted deterministic finite automata (DFAs) and weighted nondeterminizible finite automata (NFAs), and each of these have counter-free versions. Surprisingly, these two diverge in the weighted setting, despite being equivalent in the Boolean setting. We prove that, as language models, these transformers define exactly the same weighted languages as weighted counter-free DFAs.

Jerad et al. (2025) proved that *leftmost* UHATs, as language recognizers, recognize the same languages as a fragment of LTL, called in our notation $\mathsf{TL}[\mathbf{P}]$, or a subclass of counter-free DFAs called partially ordered DFAs. Similarly, Li and Cotterell (2025) proved that softmax attention transformers (SMATs) with fixed precision, as language recognizers, recognize the same class of languages. But here we show that as language models, these variants become slightly more powerful.

Yang et al. (2025) considered SMATs with fixed precision but arbitrary precision inside attention. As language recognizers, these transformers are exactly equivalent to a temporal logic extended with

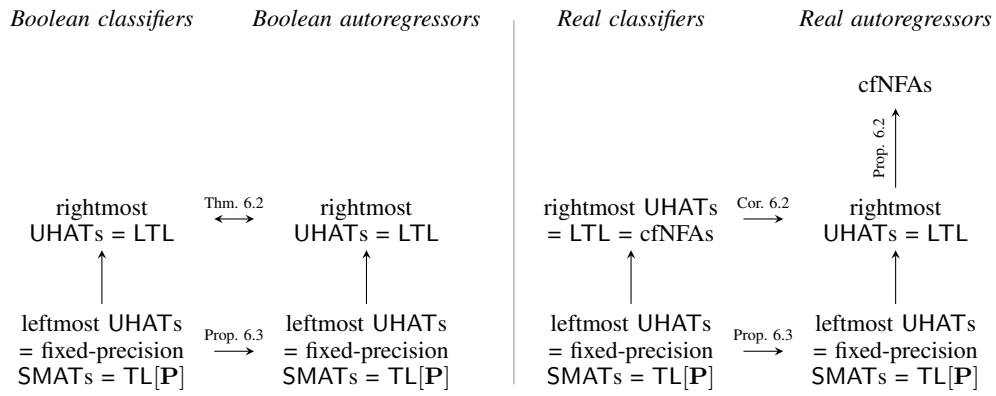

Figure 1: In the Boolean semiring, equivalences from the literature (Yang et al., 2024; Jerad et al., 2025; Yang et al., 2025) carry over from classifiers to autoregressors; however, sometimes autoregressors are more expressive than classifiers. In the real semiring, LTL and counter-free DFAs and NFAs become less expressive than counter-free NFAs, and rightmost UHATs are only as expressive as the former. Key: $\rightarrow$ = strict inclusion, $\leftrightarrow$ = equivalence.

counting operators. But here we show that, as language models, they become slightly more powerful when considering different depths.

In §3, we define notational preliminaries. We then (§4) define the classes of transformers we consider, and show how they can be used as classifiers and as autoregressors. We then (§5) introduce two other formalisms, deterministic finite automata (DFAs) and linear temporal logic (LTL), and show how they can also be seen as instantiations of the framework we showed for transformers. Then (§6), using LTL, we investigate the expressive power of transformers as both classifiers and autoregressors, yielding the results shown in Fig. 1.

2 Related Work

Theoretical study of transformers as language models has not gone totally neglected. Hahn (2020) compared a SMAT language model with a probabilistic finite automaton for parity (strings that have an odd number of 1's). Yao et al. (2021), following previous work on RNNs, consider a transformer language model to ϵ**-generate** a language if it assigns probability at least ϵ to each symbol in every string in the language (and no strings not in the language). They also discuss how to convert a construction for a bounded Dyck language (strings of matching parentheses up to a certain depth) from an ϵ-generator to a language recognizer. These studies were specialized to particular languages and used specialized ways of comparing distributions that don't generalize in an obvious way.

Svete and Cotterell (2024) made progress by showing that average-hard attention transformer language models can exactly express all n-gram language models.

Bhattamishra et al. (2020) proved theoretical results on transformers as language recognizers but carried out experiments on transformer language models for the **character prediction task**, which is to predict, at each position, the set of next possible symbols (what we will call Boolean autoregression). This experimental setup was previously used in studies of RNNs, and has been adopted in other studies of transformers (Huang et al., 2025; Yang et al., 2025). We discuss the experiment of Yang et al. (2025) in §6.6.

3 Preliminaries

Throughout this paper, we work with weighted languages. We define some key concepts here, but for a more detailed introduction, see the handbook chapter by Droste and Kuske (2021).

Let Σ be an **alphabet**, that is, a finite, non-empty set of **symbols**, and let Σ^* be the set of strings over Σ. We often augment Σ with start and end symbols BOS and EOS, but never consider BOS or EOS to belong to Σ. For any string $\boldsymbol{w} = w_1 \cdots w_n$, we write the length of $\boldsymbol{w}$ as $|\boldsymbol{w}| = n$.

We think of weights and probabilities as elements of **semirings**, an abstraction of the usual addition and multiplication operations that allows results and algorithms apply generically to multiple settings. A semiring $\mathbb{K}$ has an addition operation $\oplus$, additive identity $\mathbf{0}$, multiplication operation $\otimes$, and multiplicative identity $\mathbf{1}$. The two semirings we focus on in this paper are the **(extended) real semiring** $\overline{\mathbb{R}}_{\geq 0}$, which contains all nonnegative real numbers and $+\infty$, and in which $\oplus$ and $\otimes$ are real addition and multiplication; and the **Boolean semiring** $\mathbb{B}$, in which $\oplus$ is disjunction ($\vee$), $\mathbf{0}$ is false ($\bot$), $\otimes$ is conjunction ($\wedge$), and $\mathbf{1}$ is true ($\top$).

A **weighted language** (also called a **formal power series**) is a function $S \colon \Sigma^* \to \mathbb{K}$. If $\mathbb{K}$ is complete (that is, it allows infinite summations, as $\overline{\mathbb{R}}_{\geq 0}$ and $\mathbb{B}$ do), then we call a weighted language **normalized** if $\sum_{\boldsymbol{w} \in \Sigma^*} S(\boldsymbol{w}) = \mathbf{1}$. The **support** of a weighted language is the set of strings with nonzero weight: $\text{supp}(S) = \{\boldsymbol{w} \in \Sigma^* \mid S(\boldsymbol{w}) \neq \mathbf{0}\}$.

For sets X and Y, we write Y^X for the set of functions from X to Y, and 2^X for $\{0,1\}^X$ or the power set of X.

4 Transformer Language Models

4.1 Unique Hard Attention Transformers

Following Yang et al. (2024), we use **unique-hard attention transformers** (UHATs), specifically, with rightmost-hard attention, strict future masking, and no position embeddings. We give a definition of strictly masked rightmost-hard attention here; for a definition of the rest of the network, see, for example, the survey by Strobl et al. (2024).

The attention function receives a sequence of query vectors $\mathbf{q}^{(i)} \in \mathbb{R}^{d_k}$, key vectors $\mathbf{k}^{(j)} \in \mathbb{R}^{d_k}$, and value vectors $\mathbf{v}^{(j)} \in \mathbb{R}^d$, for $i, j \in [n]$. At each position i, it computes

$$\text{Att}\left((\mathbf{q}^{(i)})_{i \in [n]}, (\mathbf{k}^{(j)})_{j \in [n]}, (\mathbf{v}^{(j)})_{j \in [n]}\right) = (\mathbf{c}^{(i)})_{i \in [n]} \tag{1}$$

where

$$\begin{aligned}
a_i(j) &= \mathbf{q}^{(i)} \cdot \mathbf{k}^{(j)} && \text{is an attention score for each position } j, \\
a_i^* &= \max_{j < i} a_i(j) && \text{is the maximum attention score,} \\
j_i &= \max\{j < i \mid a_i(j) = a_i^*\} && \text{is the rightmost maximum-scoring position, and} \\
\mathbf{c}^{(i)} &= \begin{cases} \mathbf{v}^{(j_i)} & \text{if } i > 0 \\ \mathbf{0} & \text{if } i = 0 \end{cases} && \text{is the attention output.}
\end{aligned}$$

Given an input string $\boldsymbol{w} = w_1 \cdots w_n$, a transformer $\mathcal{T}$ prepends a symbol $w_0 = \text{BOS}$ and computes a sequence of "states" $\mathcal{T}(x) = (h_0, \ldots, h_n)$, where h_i is the state after reading w_i. There are at least two ways to use $\mathcal{T}$ to define a weighted language.[1]

4.2 Classifiers

The first way that a transformer can define a weighted language is as a **classifier**.

Definition 4.1. *A UHAT **classifier** is a UHAT $\mathcal{T} \colon \Sigma^* \to (\mathbb{R}^d)^*$ together with a function $c \colon \mathbb{R}^d \to \mathbb{K}$, which outputs a scalar weight at the last position only:*

$$S_c(\boldsymbol{w}) = c(h_n). \tag{2}$$

[1] A third intermediate way would be to multiply the weights at each position like an autoregressive model, but not to pass the output symbol at each position autoregressively to the input at the next position. Although interesting in its own right, it has not, to our knowledge, been used with any neural sequence models, and we do not explore this style of model here.

For the Boolean semiring ($\mathbb{K} = \mathbb{B}$), we accept a string iff the transformer outputs $\top$ at the last position. For example, the output function could be $c(y) = \mathbb{I}\{\mathbf{W}y + \mathbf{b} \geq 0\}$, where $\mathbf{W}$ and $\mathbf{b}$ are parameters. This is the setup used for binary classification with a transformer encoder (Devlin et al., 2019) and in most theoretical papers on transformer expressivity.

4.3 Autoregressive Models

The second way for a transformer to define a weighted language is as an **autoregressive model**, or an **autoregressor** for short (by analogy with *classifier*). We define a function $p\colon \mathbb{R}^d \to \mathbb{K}^{\Sigma \cup \{\text{EOS}\}}$, which outputs at each position a weight distribution for the next symbol, including EOS. To line up with the more familiar notation of conditional probability distributions, we write, for all $\sigma \in \Sigma \cup \{\text{EOS}\}$,

$$\Pr_p(\sigma \mid \boldsymbol{w}_{\leq i}) = p(h_i)(\sigma). \tag{3}$$

As suggested by this notation, we want $\Pr_p(\cdot \mid \boldsymbol{u})$ to be a probability distribution over $\Sigma \cup \{\text{EOS}\}$. But we impose a stronger condition. First, we extend $\Pr_p(\sigma \mid \boldsymbol{u})$ to the probability distribution of suffixes given (possibly empty) prefixes:

$$\Pr_p(\boldsymbol{v} \mid \boldsymbol{u}) = \left(\bigotimes_{i=1}^{|\boldsymbol{v}|} \Pr_p(v_i \mid \boldsymbol{u}\boldsymbol{v}_{<i}) \right) \otimes \Pr_p(\text{EOS} \mid \boldsymbol{u}\boldsymbol{v}) \tag{4}$$

$$\Pr_p(\boldsymbol{w}) = \Pr_p(\boldsymbol{w} \mid \epsilon). \tag{5}$$

Then we require that every such distribution sums to one:

Definition 4.2. *A UHAT **autoregressor** is a UHAT $\mathcal{T}\colon \Sigma^* \to (\mathbb{R}^d)^*$ together with a function $p\colon \mathbb{R}^d \to \mathbb{K}^{\Sigma \cup \{\text{EOS}\}}$ such that for all $\boldsymbol{u} \in \Sigma^*$,*

$$\bigoplus_{\boldsymbol{v} \in \Sigma^*} \Pr_p(\boldsymbol{v} \mid \boldsymbol{u}) = \mathbf{1}. \tag{6}$$

This implies that:

- The autoregressor generates strings symbol-by-symbol. That is, for all prefixes $\boldsymbol{u}$, $\sum_{\sigma \in \Sigma \cup \{\text{EOS}\}} \Pr_p(\sigma \mid \boldsymbol{u}) = \mathbf{1}$.
- Generation does not have any dead ends or endless loops. That is, for all prefixes $\boldsymbol{u}$,

$$\bigotimes_{i=1}^{n} \Pr_p(u_i \mid \boldsymbol{u}_{<i}) \neq \mathbf{0} \implies \Pr_p(\boldsymbol{u}\boldsymbol{v}) \neq \mathbf{0} \text{ for some suffix } \boldsymbol{v}.$$

In the real semiring ($\mathbb{K} = \overline{\mathbb{R}}_{\geq 0}$), a typical example of such an output function is $p(h) = \mathrm{softmax}(\mathbf{W}h + \mathbf{b})$.

5 Other Formalisms

We can analogously use other formalisms to define classifier or autoregressive models. Any state encoder which sends a string $w_1 \cdots w_n$ to a sequence of "states" $h_0, \ldots, h_n \in Q$ can be equipped with an output function $c\colon Q \to \mathbb{K}$ to give a classifier model or $p\colon Q \to \mathbb{K}^{\Sigma \cup \{\text{EOS}\}}$ to give an autoregressive model exactly as we did with transformers above.

5.1 Finite Automata

Definition 5.1 (Deterministic finite automaton)**.** *A **deterministic finite automaton** (DFA) is a tuple $\mathcal{A} = (\Sigma, Q, \delta, \iota)$, where*

- *Σ is an alphabet*
- *Q is a finite set of **states***
- *$\delta\colon Q \times \Sigma \to Q$ is a **transition function***

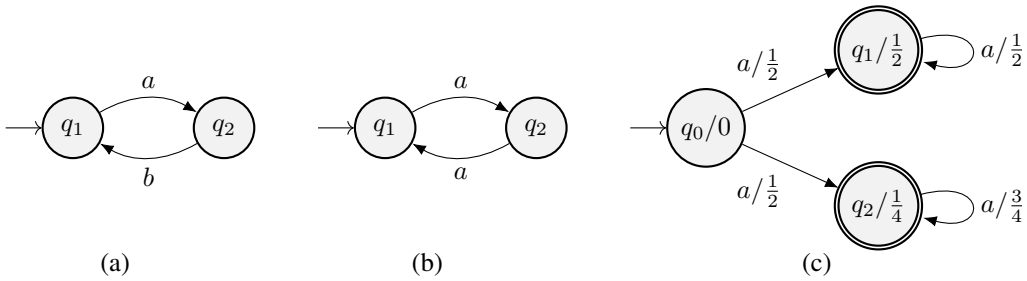

Figure 2: (a) A DFA that is counter-free (with $k = 2$). (b) A DFA that is not counter-free, because for all k, the strings a^k and a^{k+1} have opposite actions. (c) A counter-free weighted NFA that has no equivalent weighted DFA (Prop. 6.2).

- $\iota \in Q$ *is the* ***initial*** *state.*

We extend δ *to a mapping* $\delta^* \colon Q \times \Sigma^* \to Q$ *such that:*

$$\begin{aligned} \delta^*(q, \epsilon) &= q \\ \delta^*(q, \sigma \boldsymbol{w}) &= \delta^*(\delta(q, \sigma), \boldsymbol{w}). \end{aligned} \tag{7}$$

A DFA $\mathcal{A}$ defines a state encoder

$$\begin{aligned} &\mathcal{A} \colon \Sigma^* \to Q^* \\ &\mathcal{A}(\boldsymbol{w})_i = \begin{cases} \iota & i = 0 \\ \delta^*(\iota, w_1 \cdots w_i) & 0 < i \leq n. \end{cases} \end{aligned} \tag{8}$$

A DFA with classifier outputs in the Boolean semiring is the same as the standard definition of a DFA: the states that output $\top$ are the accept states, and the states that output $\bot$ are the reject states.

A DFA with autoregressive outputs in the real semiring is the same as the standard definition of a weighted DFA: when it is in state q, the next input symbol σ determines both the next state $\delta(q, \sigma)$ as well as the symbol weight $p(q)(\sigma)$. Moreover, each state has an accepting weight $p(q)(\text{EOS})$.

In this paper, we are only interested in the following subclass of finite automata called **counter-free automata**, which we abbreviate as cfDFAs.

Definition 5.2 (Counter-free automaton)**.** *We say that a DFA with transition function* δ *is* ***counter-free*** *if there exists some* k *such that for all states* q*, all strings* $\boldsymbol{w}$*, we have* $\delta^*(q, \boldsymbol{w}^k) = \delta^*(q, \boldsymbol{w}^{k+1})$.

Examples of counter-free and non-counter-free DFAs are shown in Fig. 2ab.

5.2 Linear Temporal Logic

Definition 5.3 (Linear temporal logic)**.** *The formulas of past* LTL *are defined by the grammar*

$$\begin{aligned} \phi ::= {} & \neg\phi_1 \mid \phi_1 \wedge \phi_2 \mid \text{BOS} \\ & \mid \sigma && \sigma \in \Sigma \\ & \mid \mathbf{Y}\phi_1 && \textit{Yesterday} \\ & \mid \mathbf{P}\phi_1 && \textit{Previously} \\ & \mid \mathbf{H}\phi_1 && \textit{Historically} \\ & \mid \phi_1 \, \mathbf{S} \, \phi_2 && \textit{Since} \end{aligned}$$

The semantics of formulas is given by the relation $\boldsymbol{w}, i \models \phi$ ("$\boldsymbol{w}$ satisfies ϕ at position i"), defined as follows:

$$\boldsymbol{w}, i \models \neg\phi_1 \iff \boldsymbol{w}, i \not\models \phi_1 \tag{9a}$$
$$\boldsymbol{w}, i \models \phi_1 \wedge \phi_2 \iff \boldsymbol{w}, i \models \phi_1 \text{ and } \boldsymbol{w}, i \models \phi_2 \tag{9b}$$
$$\boldsymbol{w}, i \models \text{BOS} \iff i = 0 \tag{9c}$$
$$\boldsymbol{w}, i \models \sigma \iff w_i = \sigma \tag{9d}$$
$$\boldsymbol{w}, i \models \mathbf{Y}\phi_1 \iff i > 0 \text{ and } \boldsymbol{w}, i-1 \models \phi_1 \tag{9e}$$
$$\boldsymbol{w}, i \models \mathbf{P}\phi_1 \iff \boldsymbol{w}, j \models \phi_1 \text{ for some } j \leq i \tag{9f}$$
$$\boldsymbol{w}, i \models \mathbf{H}\phi_1 \iff \boldsymbol{w}, j \models \phi_1 \text{ for all } j \leq i \tag{9g}$$
$$\boldsymbol{w}, i \models \phi_1 \,\mathbf{S}\, \phi_2 \iff (\boldsymbol{w}, j \models \phi_2 \text{ for some } j \leq i) \text{ and } (\boldsymbol{w}, j' \models \phi_1 \text{ for all } j < j' \leq i). \tag{9h}$$

We write $\boldsymbol{w} \models \phi$ as shorthand for $\boldsymbol{w}, |\boldsymbol{w}| \models \phi$.

Note that $\mathbf{H}\varphi$ is equivalent to $\neg\mathbf{P}\neg\varphi$, so only one will be necessary. For any set of operators $\mathcal{O} \subseteq \{\mathbf{Y}, \mathbf{H}, \mathbf{S}\}$, we write $\mathsf{TL}[\mathcal{O}]$ for the set of formulas using only operators in $\mathcal{O}$. Thus past $\mathsf{LTL} = \mathsf{TL}[\mathbf{Y}, \mathbf{S}]$. Given a tuple of formulas $\Phi = (\phi_1, \ldots, \phi_m)$, we can define a state encoder

$$\Phi \colon \Sigma^* \to (\mathbb{B}^m)^*$$
$$\Phi(\boldsymbol{w})_i = (\mathbb{I}\{\boldsymbol{w}, i \models \phi_1\}, \ldots \mathbb{I}\{\boldsymbol{w}, i \models \phi_m\}).$$

Droste and Gastin (2019) define a weighted first-order logic, with several variations corresponding to several subclasses of weighted counter-free automata. Mandrali and Rahonis (2013; 2015) do the same for LTL. Both of these logics have, roughly speaking, four layers: (1) a core Boolean logic, (2) weights conditioned on formulas, (3) products over positions, and (4) addition and sums over positions. This is similar to our framework, which has (1) a core Boolean logic, (2) classifier output functions that can choose weights conditioned on formulas, and (3) autoregressive output functions that can also compute products over positions.

6 Expressivity Results

Previous results have shown that UHATs, LTL, and cfDFAs are equivalent in terms of language recognition. In the weighted language setting, different model setups may lead to different levels of expressivity.

In §6.1, we use existing results on the equivalence of UHATs, LTL, and counter-free DFAs to show that these formalisms are also equivalent as weighted classifiers and as autoregressors.

Next, we compare the expressivity of classifier versus autoregressive models. Given the equivalence of the above formalisms, we will mainly discuss LTL.

In §6.2, we will show that LTL classifiers define exactly the aperiodic step functions (defined below). In the Boolean semiring, this is the same class of weighted languages that LTL autoregressors define, which is the main result of §6.3.

However, when we consider fragments of LTL, this equivalence breaks down, and autoregressors may become more expressive than classifiers (§6.5).

Moreover, in the real semiring, LTL classifiers become less expressive than LTL autoregressors, and while LTL autoregressors remain equivalent to counter-free DFAs, both are less expressive than counter-free NFAs (§6.4).

6.1 State Encoders

We say that two state encoders $\tau_1 \colon \Sigma^* \to Q_1^*$ and $\tau_1 \colon \Sigma^* \to Q_2^*$ are **equivalent** if there is a bijection $f \colon Q_1 \to Q_2$ such that for all $\boldsymbol{w} \in \Sigma^*$, $f(\tau_1(\boldsymbol{w})) = \tau_2(\boldsymbol{w})$.

Theorem 6.1. UHAT*s,* LTL*, and cfDFAs define equivalent* state encoders.

The proof is an adaptation of results from (Yang et al., 2024; Schützenberger, 1965; McNaughton and Papert, 1971; Kamp, 1968) connecting UHATs, LTL and cfDFAs as language recognizers.

Proof. See App. A. □

The following is an immediate consequence of Thm. 6.1 and the definitions of classifier and autoregressive models.

Corollary 6.1. UHAT*s,* LTL*, and counter-free DFAs as classifier models define the same weighted languages. Similarly when they are used as autoregressive models.*

Proof. By the previous theorem, all these formalisms define equivalent state encoders. Therefore there exist output functions in which they define the same weighted languages. □

6.2 CLASSIFIER MODELS

Definition 6.1. *An **aperiodic step function** (Droste and Gastin, 2008) is a weighted language* $S\colon \Sigma^* \to \mathbb{K}$ *such that* $S(\boldsymbol{w}) = \bigoplus_{i=1}^{m} k_i \otimes \mathbb{I}\{\boldsymbol{w} \in L_i\}$ *where* $L_1, \ldots, L_m$ *are aperiodic (that is, counter-free) regular languages.*

Proposition 6.1. *An* LTL *classifier defines the aperiodic step functions.*

Proof. Given any aperiodic step function as defined above, we can write, for each L_i, an LTL formula ϕ_i. Then we can write a classifier output function $c(h) = \bigoplus_{i=1}^{m} k_i \otimes h_i$.

Conversely, given an LTL classifier consisting of a tuple of formulas $(\phi_1, \ldots, \phi_m)$ and an output function $c(h)$, for every $h \in 2^{[m]}$, write the formula $\phi_h = \bigwedge_{i|h_i=1} \phi_i \wedge \bigwedge_{i|h_i=0} \neg\phi_i$. For every h, let L_h be the language defined by ϕ_h. Then the weighted language can be written as the step function $S(\boldsymbol{w}) = \bigoplus_{h \in 2^{[m]}} c(h) \otimes \mathbb{I}\{\boldsymbol{w} \in L_h\}$. □

Corollary 6.2. *In the real semiring, the weighted language* $(\frac{1}{2}a)^*$ *is expressible by an* LTL *autoregressor, but not by any* LTL *classifier.*

Proof. This language has an infinite number of string weights, but an aperiodic step function can only output a finite number of different weights. On the other hand, it is easy to write a weighted counter-free DFA that defines this weighted language. □

6.3 AUTOREGRESSIVE MODELS

In this section we will focus on $\mathbb{B}$ and $\overline{\mathbb{R}}_{\geq 0}$ weighted autoregressors – the two settings that arise in practice. We will see that LTL classifiers and LTL autoregressors are equivalent, but with an important caveat. If we consider fragments of LTL that have only a subset of the temporal operators, the equivalence only holds under certain subsets.

Theorem 6.2. *For any set of operators* $\mathcal{O} \subseteq \{\mathbf{Y}, \mathbf{H}, \mathbf{S}\}$*:*

- *(a) Let* S_1 *be a weighted language defined by a* $\mathsf{TL}[\mathcal{O}]$ *classifier. There exists a* $\overline{\mathbb{R}}_{\geq 0}$*-weighted* $\mathsf{TL}[\mathcal{O}]$ *autoregressor satisfying Eq.* (6) *defining a weighted language* S_2 *such that* $\text{supp}(S_1) = \text{supp}(S_2)$.
- *(b) Let* S_1 *be a weighted language defined by a* $\mathsf{TL}[\mathcal{O}]$ *autoregressor over a semiring with no non-trivial zero divisors (that is, whenever* $k_1 \otimes k_2 = 0 \iff k_1 = 0 \vee k_2 = 0$*). There exists a* $\overline{\mathbb{R}}_{\geq 0}$*-weighted* $\mathsf{TL}[\mathcal{O} \cup \{\mathbf{P}, \mathbf{Y}\}]$ *classifier defining a weighted language* S_2 *such that* $\text{supp}(S_1) = \text{supp}(S_2)$.

Proof. See App. B.3. □

To prove this, we need to introduce two new operators as "syntactic sugar" that do not increase the expressivity of the logic.

Lemma 6.1. *There is a transformation* next_σ *from formulas of* $\mathsf{TL}[\mathcal{O}]$ *to formulas of* $\mathsf{TL}[\mathcal{O}]$ *such that for any formula* ϕ *of* $\mathsf{TL}[\mathcal{O}]$ *and for all* $\boldsymbol{w} \in \Sigma^*$*,*

$$\boldsymbol{w} \models \text{next}_\sigma(\phi) \iff \boldsymbol{w}a \models \phi. \tag{10}$$

Intuitively, next_σ removes an a on the right; in other words, $\mathrm{next}_\sigma(\phi)$ defines the right Brzozowski derivative (Brzozowski, 1964) of the language defined by ϕ.

Proof. See App. B.1. □

Lemma 6.2. *There is a transformation* prefix *from formulas of* $\mathsf{TL}[\mathcal{O}]$ *to formulas of* $\mathsf{TL}[\mathcal{O}]$ *such that for any formula* ϕ *of* $\mathsf{TL}[\mathcal{O}]$ *and for all* $\boldsymbol{w} \in \Sigma^*$*,*

$$\boldsymbol{w} \models \mathrm{prefix}(\phi) \iff \textit{there exists } \boldsymbol{v} \textit{ such that } \boldsymbol{wv} \models \phi. \tag{11}$$

Proof. See App. B.2. □

With the above results, we can also show the following.

Corollary 6.3. *Let* ϕ *be a* $\mathsf{TL}[\mathcal{O}]$ *formula and let* $\phi' = \mathrm{prefix}(\phi)$*, whose existence is guaranteed by §6.3. Computing* ϕ' *is in general PSPACE-hard; there is an exponential-time, polynomial-space construction and* $|\phi'|$ *can be exponential in* $|\phi|$*.*

For each $\boldsymbol{u} \in \Sigma^*$*, let* $\textsc{Tails}_f(\phi', \boldsymbol{u})$ *be true iff there is a* $\boldsymbol{v} \in \Sigma^*$ *such that* $\boldsymbol{uv} \models \phi$*. Then deciding* $\textsc{Tails}_f(\phi', \boldsymbol{u})$ *is in general PSPACE-complete where the problem size is defined as* $|\phi'| + |\boldsymbol{u}|$ *.*

Proof. The algorithm from the proof of §6.3 can be implemented as a PSPACE algorithm that outputs each disjunct in the formula ϕ' one by one. Hardness refers to the problem of checking ϕ' (e.g. ϕ' is unsatisfiable). See App. B.4. □

6.4 Nondeterministic finite automata

In the unweighted setting, LTL is expressively equivalent to counter-free automata (Schützenberger, 1965; McNaughton and Papert, 1971), but in the weighted case, there are several nonequivalent analogues of counter-free automata (Droste and Gastin, 2008). Here, we are primarily concerned with the distinction between weighted counter-free DFAs and **nondeterministic finite automata** (NFAs), in which a state can have more than one outgoing transition with the same symbol (see App. C for a definition). Consequently, we cannot think of an NFA as mapping a string to a single sequence of states. To use an NFA as an autoregressive model, we have to sum the weights of many sequences of states. Although unweighted NFAs are determinizable, not all weighted NFAs are determinizable (Mohri, 1997). Likewise, not all counter-free weighted NFAs are determinizable, as the following shows.

Proposition 6.2. *As autoregressors, counter-free NFAs define more weighted languages than counter-free DFAs (or* UHAT*s or* LTL*) do.*

Fig. 2c shows an example of a counter-free weighted NFA that is not determinizable. See App. C for a proof.

6.5 Fragments of LTL

Li and Cotterell (2025) show that fixed-precision future-masked transformers are equivalent to $\mathsf{TL}[\mathbf{P}]$, and Jerad et al. (2025) show that future-masked leftmost-hard attention transformers are also equivalent to $\mathsf{TL}[\mathbf{P}]$. However, in this section we show that for autoregressors, these equivalences break.

When the set of operators $\mathcal{O}$ lacks either $\mathbf{H}$ or $\mathbf{Y}$, the asymmetry in Thm. 6.2 suggests that Boolean autoregressors are more expressive than classifiers. The following shows that this is indeed the case.

Proposition 6.3. *The language* $(ab)^*$ *is defined by a Boolean* $\mathsf{TL}[\emptyset]$ *autoregressor but not defined by any* $\mathsf{TL}[\mathbf{H}]$ *or* $\mathsf{TL}[\mathbf{Y}]$ *classifier.*

Proof. Consider the state encoder $\Phi = (\text{BOS}, a, b)$ and the output function

$$p(q_{\text{BOS}}, q_a, q_b)(a) = \top \iff q_{\text{BOS}} = \top \text{ or } q_b = \top$$
$$p(q_{\text{BOS}}, q_a, q_b)(b) = \top \iff q_a = \top$$
$$p(q_{\text{BOS}}, q_a, q_b)(\text{EOS}) = \top \iff q_{\text{BOS}} = \top \text{ or } q_b = \top$$

But a formula in $\mathsf{TL}[\mathbf{Y}]$ can't distinguish between strings that differ beyond their last k symbols (for some constant k depending on the formula), and for any k, we have $ab(ab)^{\lceil k/2 \rceil} \in (ab)^*$ but $ba(ab)^{\lceil k/2 \rceil} \notin (ab)^*$. A formula in $\mathsf{TL}[\mathbf{P}]$ (which is equivalent to $\mathsf{TL}[\mathbf{H}]$) can only define a stutter-invariant language, which is a language L such that for all $\boldsymbol{u}, \sigma, \boldsymbol{v}$, we have $\boldsymbol{u}\sigma\boldsymbol{v} \in L \iff \boldsymbol{u}\sigma\sigma\boldsymbol{v} \in L$ (Peled and Wilke, 1997). And $(ab)^*$ is not stutter-invariant, because $ab \in (ab)^*$ but $aabb \notin (ab)^*$. □

However, the expressiveness added by autoregression seems somewhat limited, as $(aab)^*$ is not definable.

We define the **$\mathbf{Y}$-depth** of a formula ϕ as the number of nested $\mathbf{Y}$ operators in ϕ.

Proposition 6.4. *$(aab)^*$ is not definable by any formula of $\mathsf{TL}[\mathbf{P}, \mathbf{Y}]$ with $\mathbf{Y}$-depth 1.*

Proof. See App. D. □

This fact implies a statement about language models in our framework.

Theorem 6.3. *$(aab)^*$ is not definable by any autoregressive leftmost UHAT language model.*

Proof. Leftmost UHATs are expressively equivalent to $\mathsf{TL}[\mathbf{P}]$ (Jerad et al., 2025). Following the construction in Thm. 6.2, a Boolean $\mathsf{TL}[\mathbf{P}]$ autoregressor can be simulated using a $\mathsf{TL}[\mathbf{P}, \mathbf{Y}]$ formula with $\mathbf{Y}$-depth 1. By Prop. 6.4, no such formula can define $(aab)^*$. □

6.6 Temporal Logic with Counting

Other formalisms besides the ones discussed above have been proposed for comparison with transformers. Yang et al. (2025) prove that SMATs, with fixed precision outside attention and arbitrary precision inside attention, are equivalent to a temporal logic with counting operators, $\mathsf{TL}[\overleftarrow{\#}, +]$. They considered the family of languages

$$L_1 = a^* \tag{12}$$
$$L_{k+1} = \begin{cases} L_k b^* & k \text{ even} \\ L_k a^* & k \text{ odd} \end{cases} \tag{13}$$

and showed that, as Boolean classifiers, transformers with depth k can recognize L_k. But their experiments were on the symbol-prediction task (§2), closely related to Boolean autoregression. They showed both theoretically and experimentally that SMATs with depth k can solve the symbol-prediction task for not only L_k, but L_{k+2}.

In the present framework, this discrepancy can be readily explained. Like $\mathsf{TL}[\mathbf{P}]$, the logic $\mathsf{TL}[\overleftarrow{\#}, +]$ lacks a $\mathbf{Y}$ operator or an equivalent. So it is more expressive as an autoregressor than as a classifier.

7 Conclusion

We have shown that theoretical results on transformers as Boolean classifiers (as most theoretical results in the literature are) sometimes carry over to real-weighted and/or autoregressive settings, but they sometimes do not. We have laid out a framework for studying other variants of transformers and other automata or logics as real-weighted autoregressors, leading to theoretical results that can make more accurate predictions about language models as used in practice.

REFERENCES

Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008. ISBN 978-0-262-02649-9. URL https://mitpress.mit.edu/9780262026499/principles-of-model-checking/.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the ability and limitations of Transformers to recognize formal languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7096–7116, 2020. doi:10.18653/v1/2020.emnlp-main.576.

J. A. Brzozowski and Faith E. Fich. Languages of $\mathcal{R}$-trivial monoids. *Journal of Computer and System Sciences*, 20(1):32–49, 1980. doi:10.1016/0022-0000(80)90003-3.

Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, October 1964. ISSN 0004-5411. doi:10.1145/321239.321249. URL https://doi.org/10.1145/321239.321249.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional Transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL HLT)*, pages 4171–4186, 2019. doi:10.18653/v1/N19-1423.

Manfred Droste and Paul Gastin. On aperiodic and star-free formal power series in partially commuting variables. *Theory of Computing Systems*, 42(4):608–631, 2008. doi:10.1007/s00224-007-9064-z.

Manfred Droste and Paul Gastin. Aperiodic weighted automata and weighted first-order logic. In *Proceedings of the 44th International Symposium on Mathematical Foundations of Computer Science*, 2019. doi:10.4230/LIPICS.MFCS.2019.76.

Manfred Droste and Dietrich Kuske. Weighted automata. In Jean-Éric Pin, editor, *Handbook of Automata Theory*, pages 113–150. European Mathematical Society Publishing House, Zürich, Switzerland, 2021. doi:10.4171/AUTOMATA-1/4.

Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of IJCAI*, pages 854–860, 2013. URL https://www.ijcai.org/Proceedings/13/Papers/132.pdf.

Michael Hahn. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171, 2020. doi:10.1162/tacl_a_00306.

Xinting Huang, Andy Yang, Satwik Bhattamishra, Yash Sarrof, Andreas Krebs, Hattie Zhou, Preetum Nakkiran, and Michael Hahn. A formal framework for understanding length generalization in transformers. In *Proceedings of the Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=U49N5V51rU.

Selim Jerad, Anej Svete, Jiaoda Li, and Ryan Cotterell. Unique hard attention: A tale of two sides. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 977–996, 2025. doi:10.18653/v1/2025.acl-short.76.

Johan Anthony Willem Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, 1968. URL https://www.proquest.com/docview/302320357.

Jiaoda Li and Ryan Cotterell. Characterizing the expressivity of transformer language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://arxiv.org/abs/2505.23623. To appear.

Eleni Mandrali and George Rahonis. Characterizations of weighted first-order logics over semirings. In *Algebraic Informatics: 5th International Conference (CAI)*, pages 247–259, 2013. doi:10.1007/978-3-642-40663-8_23.

Eleni Mandrali and George Rahonis. Weighted first-order logics over semirings. *Acta Cybernetica*, 22(2):435–483, 2015. doi:10.14232/actacyb.22.2.2015.1.

Robert McNaughton and Seymour Papert. *Counter-Free Automata*. Number 65 in M.I.T. Press Research Monographs. M.I.T. Press, 1971. ISBN 9780262130769. URL https://archive.org/details/CounterFre_00_McNa.

Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997. URL https://aclanthology.org/J97-2003/.

Doron Peled and Thomas Wilke. Stutter-invariant temporal properties are expressible without the next-time operator. *Information Processing Letters*, 63(5):243–246, 1997. doi:10.1016/S0020-0190(97)00133-6.

Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970. doi:10.1016/S0022-0000(70)80006-X.

M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8 (2):190–194, 1965. doi:10.1016/S0019-9958(65)90108-7.

Howard Straubing. Finite semigroup varieties of the form $V * D$. *Journal of Pure and Applied Algebra*, 36:53–94, 1985. doi:10.1016/0022-4049(85)90062-3.

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What formal languages can transformers express? A survey. *Transactions of the Association for Computational Linguistics*, 12:543–561, 2024. doi:10.1162/tacl_a_00663.

Anej Svete and Ryan Cotterell. Transformers can represent n-gram language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 6845–6881, 2024. doi:10.18653/v1/2024.naacl-long.381.

Denis Thérien and Thomas Wilke. Temporal logic and semidirect products: An effective characterization of the until hierarchy. *SIAM Journal on Computing*, 31(3):777–798, 2001. doi:10.1137/S0097539797322772.

Andy Yang, David Chiang, and Dana Angluin. Masked hard-attention transformers recognize exactly the star-free languages. In *Advances in Neural Information Processing Systems*, volume 37, pages 10202–10235, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/13d7f172259b11b230cc5da8768abc5f-Abstract-Conference.html.

Andy Yang, Michaël Cadilhac, and David Chiang. Knee-deep in C-RASP: A transformer depth hierarchy. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://arxiv.org/abs/2506.16055. To appear.

Shunyu Yao, Binghui Peng, Christos Papadimitriou, and Karthik Narasimhan. Self-attention networks can process bounded hierarchical languages. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pages 3770–3785, 2021. doi:10.18653/v1/2021.acl-long.292.

A Equivalence of State Encoders

Theorem 6.1. UHAT*s,* LTL*, and cfDFAs define equivalent* state encoders*.*

Proof of Thm. 6.1. First we show the equivalence of state sequences defined by UHATs and LTL, and then equivalence of LTL and cfDFAs.

The essential observation (Yang et al., 2024, Lemma 22) is that the output at every position of every UHAT layer comes from a finite set $Q \subseteq \mathbb{R}^d$. So we can think of a UHAT as a function $\mathcal{T}\colon \Sigma^* \to Q^*$. For each $h \in Q$, we can construct an LTL formula ϕ_h such that $\mathcal{T}(\boldsymbol{w})_i = h \iff \boldsymbol{w}, i \models \phi_h$ (Yang et al., 2024, Theorems 2, 4). So there exists a tuple of LTL formulas $(\phi_h)_{h\in Q}$ that defines a state encoder equivalent to $\mathcal{T}$. Note that the state outputted by $\mathcal{T}$ on the prepended BOS symbol can be simulated using a BOS formula in the tuple.

In the other direction, for every tuple of LTL formulas $(\phi_1, \phi_2, \ldots, \phi_m)$ defining a state encoder $\Sigma^* \to \mathbb{B}^m$, there exists a UHAT $\mathcal{T}\colon \Sigma^* \to (\mathbb{R}^d)^*$ defining an equivalent state encoder. For each ϕ_k, we construct a transformer $\mathcal{T}_k$ which outputs $\frac{1}{2}$ if $\boldsymbol{w}, i \models \phi_k$ and $-\frac{1}{2}$ otherwise (Yang et al., 2024, Theorems 1, 3). Then we can parallel-compose all the $\mathcal{T}_k$ into a single $\mathcal{T}$ (Yang et al., 2024, Lemma 25), and add an additional layer which projects the output dimensions of each $\mathcal{T}_k$ into a single output vector $\mathbb{R}^m$ such that $\mathcal{T}(\boldsymbol{w})_i = \mathbf{e}_k \iff \boldsymbol{w}, i \models \phi_k$.

The equivalence between LTL and cfDFAS can be described a little more succinctly. Given a DFA $\mathcal{A} = (\Sigma, Q, \delta, \iota)$, for each state $q \in Q$ there exists a formula ϕ_q such that $\boldsymbol{w} \models \phi_q \iff \delta(\iota, \boldsymbol{w}) = q$, due to the expressive equivalence of LTL and cfDFAS (Schützenberger, 1965; McNaughton and Papert, 1971; Kamp, 1968). The tuple $(\phi_q)_{q\in Q}$ then defines a state encoder equivalent to $\mathcal{A}$. In the other direction, given a tuple of LTL formulas $(\phi_1, \ldots, \phi_m)$, for each $k \in [m]$ there is an automaton $\mathcal{A}_k$ that recognizes the same language as ϕ_k. Then the Cartesian product of all the $\mathcal{A}_k$ defines a state encoder equivalent to $(\phi_1, \ldots, \phi_m)$. □

B Autoregressive Model Proofs

B.1 Proof of Lem. 6.1

Lemma 6.1. *There is a transformation* next_σ *from formulas of* $\mathsf{TL}[\mathcal{O}]$ *to formulas of* $\mathsf{TL}[\mathcal{O}]$ *such that for any formula* ϕ *of* $\mathsf{TL}[\mathcal{O}]$ *and for all* $\boldsymbol{w} \in \Sigma^*$*,*

$$\boldsymbol{w} \models \mathrm{next}_\sigma(\phi) \iff \boldsymbol{w}a \models \phi. \tag{10}$$

We define next_σ recursively:

$$\mathrm{next}_\sigma(\sigma) = \top \tag{14a}$$
$$\mathrm{next}_\sigma(\sigma') = \bot \qquad \text{if } \sigma' \neq \sigma \tag{14b}$$
$$\mathrm{next}_\sigma(\textsc{bos}) = \bot \tag{14c}$$
$$\mathrm{next}_\sigma(\neg\phi) = \neg\mathrm{next}_\sigma(\phi) \tag{14d}$$
$$\mathrm{next}_\sigma(\phi_1 \wedge \phi_2) = \mathrm{next}_\sigma(\phi_1) \wedge \mathrm{next}_\sigma(\phi_2) \tag{14e}$$
$$\mathrm{next}_\sigma(\mathbf{Y}\phi) = \phi \tag{14f}$$
$$\mathrm{next}_\sigma(\mathbf{H}\phi) = \mathbf{H}\phi \wedge \mathrm{next}_\sigma(\phi) \tag{14g}$$
$$\mathrm{next}_\sigma(\phi_1 \,\mathbf{S}\, \phi_2) = (\mathrm{next}_\sigma(\phi_1) \wedge (\phi_1 \,\mathbf{S}\, \phi_2)) \vee \mathrm{next}_\sigma(\phi_2). \tag{14h}$$

Next, we prove that $\mathrm{next}_\sigma(\phi)$ satisfies Eq. (10) by induction on the structure of ϕ.

Base Cases. If $\phi = \sigma$:

$$\boldsymbol{w}, i \models \mathrm{next}_\sigma(\sigma) \overset{(14a)}{\iff} \boldsymbol{w} \models \top \tag{15a}$$
$$\overset{(9d)}{\iff} \boldsymbol{w}\sigma \models \sigma. \tag{15b}$$

If $\phi = \sigma'$ for $\sigma' \neq \sigma$:

$$\boldsymbol{w} \models \mathrm{next}_\sigma(\sigma') \overset{(14b)}{\iff} \boldsymbol{w} \models \bot \tag{15c}$$
$$\overset{(9d)}{\iff} \boldsymbol{w}\sigma \models \sigma'. \tag{15d}$$

Similarly, if $\phi = \textsc{bos}$:

$$\boldsymbol{w} \models \text{next}_\sigma(\textsc{bos}) \overset{\text{(14c)}}{\iff} \boldsymbol{w} \models \bot \tag{15e}$$
$$\overset{\text{(9c)}}{\iff} \boldsymbol{w}\sigma \models \textsc{bos}. \tag{15f}$$

Inductive Cases. If $\phi = \neg\phi_1$:

$$\boldsymbol{w} \models \text{next}_\sigma(\neg\phi_1) \overset{\text{(14d)}}{\iff} \boldsymbol{w} \models \neg\text{next}_\sigma(\phi_1) \tag{16a}$$
$$\overset{\text{(9a)}}{\iff} \boldsymbol{w} \not\models \text{next}_\sigma(\phi_1) \tag{16b}$$
$$\overset{\text{ind. hyp.}}{\iff} \boldsymbol{w}\sigma \not\models \phi_1 \tag{16c}$$
$$\overset{\text{(9a)}}{\iff} \boldsymbol{w}\sigma \models \neg\phi_1. \tag{16d}$$

If $\phi = \phi_1 \wedge \phi_2$:

$$\boldsymbol{w} \models \text{next}_\sigma(\phi_1 \wedge \phi_2) \overset{\text{(14e)}}{\iff} \boldsymbol{w} \models \text{next}_\sigma(\phi_1) \wedge \text{next}_\sigma(\phi_2) \tag{16e}$$
$$\overset{\text{(9b)}}{\iff} (\boldsymbol{w} \models \text{next}_\sigma(\phi_1)) \wedge (\boldsymbol{w} \models \text{next}_\sigma(\phi_2)) \tag{16f}$$
$$\overset{\text{ind. hyp.}}{\iff} (\boldsymbol{w}\sigma \models \phi_1) \wedge (\boldsymbol{w}\sigma \models \phi_2) \tag{16g}$$
$$\overset{\text{(9b)}}{\iff} \boldsymbol{w}\sigma \models \phi_1 \wedge \phi_2. \tag{16h}$$

If $\phi = \mathbf{Y}\phi_1$:

$$\boldsymbol{w} \models \text{next}_\sigma(\mathbf{Y}\phi_1) \overset{\text{(14f)}}{\iff} \boldsymbol{w} \models \phi_1 \tag{16i}$$
$$\overset{\text{(9e)}}{\iff} \boldsymbol{w}\sigma \models \mathbf{Y}\phi_1. \tag{16j}$$

If $\phi = \mathbf{H}\phi_1$:

$$\boldsymbol{w} \models \text{next}_\sigma(\mathbf{H}\phi_1) \overset{\text{(14g)}}{\iff} \boldsymbol{w} \models \mathbf{H}\phi_1 \wedge \text{next}_\sigma(\phi) \tag{16k}$$
$$\overset{\text{(9b)}}{\iff} (\boldsymbol{w} \models \mathbf{H}\phi_1) \wedge (\boldsymbol{w} \models \text{next}_\sigma(\phi)) \tag{16l}$$
$$\overset{\text{ind. hyp.}}{\iff} (\boldsymbol{w} \models \mathbf{H}\phi_1) \wedge (\boldsymbol{w}\sigma \models \phi_1) \tag{16m}$$
$$\overset{\text{(9g)}}{\iff} \boldsymbol{w}\sigma \models \mathbf{H}\phi_1. \tag{16n}$$

If $\phi = \phi_1 \mathbf{S} \phi_2$:

$$\boldsymbol{w} \models \text{next}_\sigma(\phi_1 \mathbf{S} \phi_2)$$
$$\overset{\text{(14h)}}{\iff} \boldsymbol{w} \models (\text{next}_\sigma(\phi_1) \wedge (\phi_1 \mathbf{S} \phi_2)) \vee \text{next}_\sigma(\phi_2) \tag{16o}$$
$$\overset{\text{(9a) and (9b)}}{\iff} (\boldsymbol{w} \models \text{next}_\sigma(\phi_1) \wedge (\boldsymbol{w} \models \phi_1 \mathbf{S} \phi_2)) \vee (\boldsymbol{w} \models \text{next}_\sigma(\phi_2)) \tag{16p}$$
$$\overset{\text{ind. hyp.}}{\iff} ((\boldsymbol{w}\sigma \models \phi_1) \wedge (\boldsymbol{w} \models \phi_1 \mathbf{S} \phi_2)) \vee (\boldsymbol{w}\sigma \models \phi_2) \tag{16q}$$
$$\overset{\text{(9h)}}{\iff} \boldsymbol{w}\sigma \models \phi_1 \mathbf{S} \phi_2. \tag{16r}$$

B.2 Proof of §6.3

Lemma 6.2. *There is a transformation* prefix *from formulas of* $\mathsf{TL}[\mathcal{O}]$ *to formulas of* $\mathsf{TL}[\mathcal{O}]$ *such that for any formula* ϕ *of* $\mathsf{TL}[\mathcal{O}]$ *and for all* $\boldsymbol{w} \in \Sigma^*$*,*

$$\boldsymbol{w} \models \text{prefix}(\phi) \iff \textit{there exists } \boldsymbol{v} \textit{ such that } \boldsymbol{wv} \models \phi. \tag{11}$$

Given a formula ϕ of $\mathsf{TL}[\mathcal{O}]$, let $\text{cl}(\phi)$ be the set of all subformulas of ϕ (including ϕ itself). Construct a DFA $M_\phi = (2^{\text{cl}(\phi)}, \Sigma, \delta, \iota, F)$, where

$$\iota = \{\chi \in \text{cl}(\phi) \mid \epsilon \models \chi\}$$
$$F = \{\Psi \subseteq \text{cl}(\phi) \mid \phi \in \Psi\}$$
$$\delta(\Psi, \sigma) = \{\chi \in \text{cl}(\phi) \mid \Psi \xrightarrow{\sigma} \chi\}$$

where the relation $\Psi \xrightarrow{\sigma} \chi$, which intuitively means that if a string $\boldsymbol{w}$ satisfies exactly the formulas in Ψ, then $\boldsymbol{w}\sigma$ satisfies χ, is defined as follows:

$$\Psi \xrightarrow{\sigma} \sigma' \text{ iff } \sigma = \sigma' \tag{17a}$$
$$\Psi \xrightarrow{\sigma} \chi_1 \wedge \chi_2 \text{ iff } \Psi \xrightarrow{\sigma} \chi_1 \text{ and } \Psi \xrightarrow{\sigma} \chi_2 \tag{17b}$$
$$\Psi \xrightarrow{\sigma} \neg\chi \text{ iff not } \Psi \xrightarrow{\sigma} \chi \tag{17c}$$
$$\Psi \xrightarrow{\sigma} \mathbf{Y}\chi \text{ iff } \chi \in \Psi \tag{17d}$$
$$\Psi \xrightarrow{\sigma} \mathbf{H}\chi \text{ iff } \mathbf{H}\chi \in \Psi \text{ and } \Psi \xrightarrow{\sigma} \chi \tag{17e}$$
$$\Psi \xrightarrow{\sigma} \chi_1 \mathbf{S} \chi_2 \text{ iff } (\chi_1 \mathbf{S} \chi_2 \in \Psi \text{ and } \Psi \xrightarrow{\sigma} \chi_1) \text{ or } \Psi \xrightarrow{\sigma} \chi_2. \tag{17f}$$

Claim B.1. *For any* $\boldsymbol{w} \in \Sigma^*$*, if* $\Psi = \{\chi \in \mathrm{cl}(\phi) \mid \boldsymbol{w} \models \chi\}$*, then* $\Psi \xrightarrow{\sigma} \chi \iff \boldsymbol{w}\sigma \models \chi$.

Proof. By induction on the structure of χ. This is all just unwinding definitions. Note by definition that $\chi \in \Psi \iff \boldsymbol{w} \models \chi$.

$$
\begin{aligned}
\Psi \xrightarrow{\sigma} \sigma' &\overset{\text{(17a)}}{\iff} \sigma = \sigma' \\
&\overset{\text{(9d)}}{\iff} \boldsymbol{w}\sigma \models \sigma'. \\
\Psi \xrightarrow{\sigma} \chi_1 \wedge \chi_2 &\overset{\text{(17b)}}{\iff} \Psi \xrightarrow{\sigma} \chi_1 \text{ and } \Psi \xrightarrow{\sigma} \chi_2 \\
&\overset{\text{ind. hyp.}}{\iff} \boldsymbol{w}\sigma \models \chi_1 \text{ and } \boldsymbol{w}\sigma \models \chi_2 \\
&\overset{\text{(9b)}}{\iff} \boldsymbol{w}\sigma \models \chi_1 \wedge \chi_2. \\
\Psi \xrightarrow{\sigma} \neg\chi &\overset{\text{(17c)}}{\iff} \text{not } \Psi \xrightarrow{\sigma} \chi \\
&\overset{\text{ind. hyp.}}{\iff} \text{not } \boldsymbol{w}\sigma \models \chi \\
&\overset{\text{(9a)}}{\iff} \boldsymbol{w}\sigma \models \neg\chi. \\
\Psi \xrightarrow{\sigma} \mathbf{Y}\chi &\overset{\text{(17d)}}{\iff} \chi \in \Psi \\
&\iff \boldsymbol{w} \models \chi \\
&\overset{\text{(9e)}}{\iff} \boldsymbol{w}\sigma \models \mathbf{Y}\chi. \\
\Psi \xrightarrow{\sigma} \mathbf{P}\chi &\overset{\text{(17e)}}{\iff} \mathbf{H}\chi \in \Psi \text{ and } \Psi \xrightarrow{\sigma} \chi \\
&\overset{\text{ind. hyp.}}{\iff} \boldsymbol{w} \models \mathbf{H}\chi \text{ and } \boldsymbol{w}\sigma \models \chi \\
&\overset{\text{(9g)}}{\iff} \boldsymbol{w}\sigma \models \mathbf{H}. \\
\Psi \xrightarrow{\sigma} \chi_1 \,\mathbf{S}\, \chi_2 &\overset{\text{(17f)}}{\iff} (\chi_1 \,\mathbf{S}\, \chi_2 \in \Psi \text{ and } \Psi \xrightarrow{\sigma} \chi_1) \text{ or } \Psi \xrightarrow{\sigma} \chi_2 \\
&\overset{\text{ind. hyp.}}{\iff} (\boldsymbol{w} \models \chi_1 \,\mathbf{S}\, \chi_2 \text{ and } \boldsymbol{w}\sigma \models \chi_1) \text{ or } \boldsymbol{w}\sigma \models \chi_2 \\
&\overset{\text{(9h)}}{\iff} \boldsymbol{w}\sigma \models \chi_1 \,\mathbf{S}\, \chi_2.
\end{aligned}
$$

□

Claim B.2. *For any* $\boldsymbol{w}$*,* $\delta(\iota, \boldsymbol{w}) = \{\chi \in \mathrm{cl}(\phi) \mid \boldsymbol{w} \models \chi\}$.

Proof. By induction on the length of $\boldsymbol{w}$.

Base case: $\delta(\iota, \epsilon) = \iota = \{\chi \mid \epsilon \models \chi\}$.

Inductive step: Assume that $\delta(\iota, \boldsymbol{w}) = \{\chi \mid \boldsymbol{w} \models \chi\} = \Psi$. Then

$$
\begin{aligned}
\delta(\iota, \boldsymbol{w}) &= \delta(\delta(\iota, \boldsymbol{w}), \sigma) \\
&= \delta(\Psi, \sigma) \\
&= \{\chi \mid \Psi \xrightarrow{\sigma} \chi\} \\
&= \{\chi \mid \boldsymbol{w}\sigma \models \chi\}.
\end{aligned}
$$

□

Claim B.3. *M_ϕ defines the same language as ϕ.*

Proof. $\delta(\iota, \boldsymbol{w}) \in F$ if and only if $\phi \in \{\chi \mid \boldsymbol{w} \models \chi\}$ if and only if $\boldsymbol{w} \models \phi$. □

Then make every co-accessible state (every state that has a path to an accept state) into an accept state. Call this new DFA M'_ϕ with accept states F'. This DFA recognizes the prefix language of M_ϕ. Finally, construct the formula

$$
\mathrm{prefix}(\phi) = \bigvee_{\Psi \in F'} \left(\bigwedge_{\chi \in \Psi} \chi \wedge \bigwedge_{\chi \in \mathrm{cl}(\phi) \setminus \Psi} \neg\chi \right).
$$

Claim B.4. *The formula* $\mathrm{prefix}(\phi)$ *defines the same language as* M'_ϕ.

Proof. Since we only changed non-accept states to accept states, Clm. B.2 still applies to M'_ϕ and ϕ.

$$\begin{aligned}
\boldsymbol{w} \in \mathcal{L}(M'_\phi) &\iff \delta(\iota, \boldsymbol{w}) \in F' \\
&\iff \{\chi \in \mathrm{cl}(\phi) \mid \boldsymbol{w} \models \chi\} \in F' && \text{Clm. B.2} \\
&\iff \text{for some } \Psi \in F',\ \chi \in \Psi \text{ iff } \boldsymbol{w} \models \chi \\
&\iff \text{for some } \Psi \in F',\ \boldsymbol{w} \models \bigwedge_{\chi \in \Psi} \chi \wedge \bigwedge_{\chi \in \mathrm{cl}(\phi) \setminus \Psi} \neg\chi \\
&\iff \boldsymbol{w} \models \bigvee_{\Psi \in F'} \left(\bigwedge_{\chi \in \Psi} \chi \wedge \bigwedge_{\chi \in \mathrm{cl}(\phi) \setminus \Psi} \neg\chi \right).
\end{aligned}$$

□

B.3 Relationship Between Classifiers and Autoregressors

Theorem 6.2. *For any set of operators $\mathcal{O} \subseteq \{\mathbf{Y}, \mathbf{H}, \mathbf{S}\}$:*

- *(a) Let S_1 be a weighted language defined by a $\mathsf{TL}[\mathcal{O}]$ classifier. There exists a $\overline{\mathbb{R}}_{\geq 0}$-weighted $\mathsf{TL}[\mathcal{O}]$ autoregressor satisfying Eq. (6) defining a weighted language S_2 such that $\mathrm{supp}(S_1) = \mathrm{supp}(S_2)$.*
- *(b) Let S_1 be a weighted language defined by a $\mathsf{TL}[\mathcal{O}]$ autoregressor over a semiring with no non-trivial zero divisors (that is, whenever $k_1 \otimes k_2 = 0 \iff k_1 = 0 \vee k_2 = 0$). There exists a $\overline{\mathbb{R}}_{\geq 0}$-weighted $\mathsf{TL}[\mathcal{O} \cup \{\mathbf{P}, \mathbf{Y}\}]$ classifier defining a weighted language S_2 such that $\mathrm{supp}(S_1) = \mathrm{supp}(S_2)$.*

Proof of Thm. 6.2. (a) Any $\mathsf{TL}[\mathcal{O}]$ classifier S_1 is defined using a tuple of formulas $\Phi = (\phi_1, \ldots, \phi_m)$ and an output function $c\colon \mathbb{B}^m \to \mathbb{K}$. We will show the case where $\Phi = (\phi)$ is equivalent to a single formula ϕ of $\mathsf{TL}[\mathcal{O}]$, $c\colon \mathbb{B} \to \mathbb{K}$, and $S_1(\boldsymbol{w}) \neq 0 \iff \boldsymbol{w} \models \phi$ (which is the most natural case) – the generalization to tuples of formulas with different output functions is straightforward.

For each $\sigma \in \Sigma$, define

$$\phi_\sigma = \mathrm{next}_\sigma(\mathrm{prefix}(\phi)) \tag{18}$$

and also define

$$\phi_{\text{EOS}} = \phi. \tag{19}$$

Then the tuple of formulas $\Phi' = (\phi_\sigma)_{\sigma \in \Sigma \cup \{\text{EOS}\}}$ defines a state encoder. We define the output function $p\colon \mathbb{B}^{|\Sigma|+1} \to \mathbb{K}$ given by

$$p(b_{\sigma_1}, \ldots, b_{\sigma_{|\Sigma|}}, b_{\text{EOS}})(\sigma) = \begin{cases} \frac{1}{|\{b_\sigma | b_\sigma = \top\}|} & b_\sigma = \top \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

Let $S_2 = p \circ \Phi'$. First, we note by construction that this autoregressor only assigns nonzero weight to strings which satisfy ϕ, so $\mathrm{supp}(S_2) = \mathrm{supp}(S_1)$. Next, we verify that this S_2 satisfies Eq. (6). First, by the defintion of the output function in Eq. (20) we know that the next-symbol probabilities sum to 1. By a standard construction (Baier and Katoen, 2008, Sec 10.10), this induces a probability distribution over the finite strings – thus $\bigoplus_{\boldsymbol{v} \in \Sigma^*} \Pr_p(\boldsymbol{v} \mid \boldsymbol{u}) = 1$ as desired.

□

Proof of Thm. 6.2. (b) Let S_1 be an autoregessor. Like before, for brevity we assume the underlying state encoder is $\Phi = (\phi)$ and the output function is $p\colon \mathbb{B} \to \mathbb{K}$ such that $S_1(\boldsymbol{w}) \neq 0 \iff \boldsymbol{w} \models \phi$. In such an autoregressor, $S_1(\boldsymbol{w}) \neq 0$ iff $S_1(\boldsymbol{w}_1 \boldsymbol{w}_2 \cdots \boldsymbol{w}_i) \neq 0$ for all $1 \leq i \leq |\boldsymbol{w}|$.

Now for σ in $\Sigma \cup \{\text{EOS}\}$ we define the formula ϕ_σ, which depends on the boolean values α and β.

$$\alpha_\sigma^\top = \mathbb{I}\{p(\top)(\sigma) \neq 0\}$$
$$\alpha_\sigma^\bot = \mathbb{I}\{p(\bot)(\sigma) \neq 0\}$$
$$\beta = \mathbb{I}\{\boldsymbol{w}, 0 \models \phi\}$$
$$\phi_\sigma = (\mathbf{Y}\phi \wedge \alpha_\sigma^\top) \vee (\neg\mathbf{Y}\phi \wedge \alpha_\sigma^\bot) \vee (\mathbf{Y}\text{BOS} \wedge ((\beta \wedge \alpha_\sigma^\top) \vee (\neg\beta \wedge \alpha_\sigma^\bot)))$$

Intuitively, α_σ is used to check if a given state assigns nonzero probability to next-symbol σ, and $\mathbf{Y}\phi$ is used to check the state at the previous position. Also, β is used to detect the state of ϕ at position 0, which my vary (by our definitions, $\beta = \top$ iff $\phi \equiv \text{BOS}$). Then, define

$$\phi' = \mathbf{H}\left(\bigvee_{\sigma \in \Sigma \cup \{\text{EOS}\}} \sigma \wedge \phi_\sigma\right)$$

Let $\Phi' = (\phi')$ and $c(b) = \top \iff b = \top$. Then $S_2 = (c \circ \Phi')$ has the same support as S_1. □

B.4 Proof of Cor. 6.3

Corollary 6.3. *Let ϕ be a $\mathsf{TL}[\mathcal{O}]$ formula and let $\phi' = \text{prefix}(\phi)$, whose existence is guaranteed by §6.3. Computing ϕ' is in general PSPACE-hard; there is an exponential-time, polynomial-space construction and $|\phi'|$ can be exponential in $|\phi|$.*

For each $\boldsymbol{u} \in \Sigma^*$, let $\text{TAILS}_f(\phi', \boldsymbol{u})$ be true iff there is a $\boldsymbol{v} \in \Sigma^*$ such that $\boldsymbol{uv} \models \phi$. Then deciding $\text{TAILS}_f(\phi', \boldsymbol{u})$ is in general PSPACE-complete where the problem size is defined as $|\phi'| + |\boldsymbol{u}|$.

Proof. To show hardness, with $\boldsymbol{w} = \epsilon$, computing $\text{TAILS}_f(\phi, \epsilon)$ is precisely finite-trace satisfiability for past LTL with $\mathbf{S}$, which is PSPACE-complete(Giacomo and Vardi, 2013). Thus the problem is PSPACE-hard whenever $\mathbf{S} \in \mathcal{O}$.

To show membership, let $\text{cl}(\phi)$ be the set of subformulas of ϕ and set $m = O(|\phi| + |\log(|\boldsymbol{w}|)|)$. Let $M_\phi = (2^{\text{cl}(\phi)}, \Sigma, \delta, \iota, F)$ be the DFA constructed from ϕ by the progression clauses in §B.2, with

$$\iota = \{\chi \in \text{cl}(\phi) \mid \epsilon \models \chi\}, \qquad F = \{\Psi \subseteq \text{cl}(\phi) \mid \phi \in \Psi\},$$

and δ as in §B.2. By Clms. B.2 and B.3,

$$\delta^*(\iota, w) = \{\chi \in \text{cl}(\phi) \mid w, |w| \models \chi\} \quad \text{and} \quad w \models \phi \iff \delta^*(\iota, w) \in F.$$

Compute $\Psi_{\boldsymbol{w}} \coloneqq \delta^*(\iota, \boldsymbol{u})$ by iteratively applying δ along $\boldsymbol{u}$: start from ι and update $\Psi \leftarrow \delta(\Psi, a)$ for each a in $\boldsymbol{u}$. This uses $O(m)$ space.

$\text{TAILS}_f(\phi, \boldsymbol{u})$ holds iff some state in F is reachable from $\Psi_{\boldsymbol{u}}$ in the graph of M_ϕ. This graph has at most 2^m nodes, where each node can be represented in $O(m)$ size and that checking whether (v, v') is an edge in the graph can be done in time (and therefore also space) polynomial in m. This immediately gives us a nondeterministic algorithm for this graph reachability that uses only $O(m)$ space, and, by Savitch's theorem (Savitch, 1970), it follows that the problem is in PSPACE.

□

C Nondeterministic Finite Automata

We give a single definition of weighted NFAs instead of factoring them into unweighted NFAs and autoregressive output functions.

Definition C.1 (Weighted Nondeterministic Finite Automaton)**.** *A **weighted nondeterministic finite automaton** is a tuple $\mathcal{A} = (\Sigma, Q, \delta, \alpha, \omega)$, where*

- *Σ is an alphabet*
- *Q is a finite set of **states***
- *$\delta\colon Q \times \Sigma \times Q \to \mathbb{K}$ is a **transition function***
- *$\iota \in Q$ is the **initial** state*

- $\omega \colon Q \to \mathbb{K}$ *is an ending weight.*

We extend δ *to* $\delta^* \colon Q \times \Sigma^* \times Q \to \mathbb{K}$*:*

$$\begin{aligned} \delta^*(q, \epsilon, q) &= \mathbf{1} \\ \delta^*(q, \epsilon, q') &= \mathbf{0} \qquad q \neq q' \\ \delta^*(q_1, \sigma \boldsymbol{w}, q_2) &= \bigoplus_{q \in Q} \delta(q_1, \sigma, q) \otimes \delta^*(q, \boldsymbol{w}, q_2). \end{aligned}$$

Then $\mathcal{A}$ *accepts* $\boldsymbol{w}$ *with weight* k *iff*

$$k = \bigoplus_{q_2 \in Q} \delta^*(\iota, \boldsymbol{w}, q_2) \otimes \omega(q_2).$$

Definition C.2. *We say that an NFA with transition function* δ *is* ***counter-free*** *if there exists some* k *such that for all states* q_1, q_2*, all strings* $\boldsymbol{w}$*, we have* $\delta^*(q_1, \boldsymbol{w}^k, q_2) = \delta^*(q_1, \boldsymbol{w}^{k+1}, q_2)$*.*

A weighted automaton is determinizable if every every pair of states which are *siblings* (can be reached by the same string) are also *twins* (all cycles by the same string have the same weight) (Mohri, 1997). See Fig. 2c. This automaton is counter-free because for any pair of states q_1, q_2 and any string $\boldsymbol{w}$, we have that $\delta^*(q_1, \boldsymbol{w}, q_2) = \delta^*(q_1, \boldsymbol{w}^2, q_2)$. However, it is not determinizable because q_1 and q_2 are siblings (both reachable by a) but not twins (the a-labeled cycles $q_1 \xrightarrow{a/\frac{1}{2}} q_1$ and $q_2 \xrightarrow{a/\frac{3}{4}} q_2$ on the two states have different weights).

D INEXPRESSIVITY OF $(aab)^*$

Proposition 6.4. $(aab)^*$ *is not definable by any formula of* $\mathsf{TL}[\mathbf{P}, \mathbf{Y}]$ *with* $\mathbf{Y}$*-depth* 1*.*

In order to prove this, we will use some results in algebraic formal language theory. The proof relies on the definition of varieties of semigroups ($\mathbf{V}$) and substitutions of formulas ($\Phi \circ \Psi$), which we will not define here (they can be found in the citations). The following is rephrasing of Proposition 5.1 shown by Straubing (1985) but specifically for $\mathbf{V} * \mathbf{LI}_2$.

Proposition D.1. *Let* $\mathbf{V}$ *be a variety of finite monoids.* $M \in \mathbf{V} * \mathbf{LI}_2$ *iff there exists a surjective morphism* $\varphi \colon \Sigma^* \to M$ *and a morphism* $\psi \colon (\Sigma^2 \times \Sigma \times \Sigma^2)^* \to N$ *where* $N \in \mathbf{V}$*, such that the following conditions hold*

- *Whenever* π_1, π_2 *are paths in* $(\Sigma^2 \times \Sigma \times \Sigma^2)^*$ *such that* $\mathcal{L}(\pi_1) = \mathcal{L}(\pi_2)$*,* $\mathcal{R}(\pi_1) = \mathcal{R}(\pi_2)$*, and* $\psi(\pi_1) = \psi(\pi_2)$*, then* $\mathcal{V}(\pi_1) = \mathcal{V}(\pi_2)$
- *Whenever* π *is a path in* $(\Sigma^2 \times \Sigma \times \Sigma^2)^*$ *such that* $\mathcal{L}(\pi) = \mathcal{R}(\pi) = w$ *and* $\psi(\pi) = \psi(\epsilon)$*, then* $\mathcal{V}(\pi) = \varphi(w)$*.*

Where a path $\pi \in (\Sigma^2 \times \Sigma \times \Sigma^2)^*$ *is of the form* $(\boldsymbol{w}_0, \sigma_1, \boldsymbol{w}_1)(\boldsymbol{w}_1, \sigma_2, \boldsymbol{w}_2) \cdots (\boldsymbol{w}_{k-2}, \sigma_k, \boldsymbol{w}_k)$ *such that* $\boldsymbol{w}_i \sigma_1 \in \Sigma \boldsymbol{w}_{i+1}$ *for all* i*. Furthermore we use the notation* $\mathcal{L}(w) = \boldsymbol{w}_0$*,* $\mathcal{R}(w) = \boldsymbol{w}_k$*, and* $\mathcal{V}(w) = \boldsymbol{w}_0 \sigma_1 \sigma_2 \cdots \sigma_{k-1} \sigma_k$*.*

Now, we prove Prop. 6.4.

Proof of Prop. 6.4. By substitution rules, each of these formulas is definable in $\mathsf{TL}[\mathbf{H}] \circ \mathsf{TL}[\mathbf{Y}]_1$. Then, the languages definable in $\mathsf{TL}[\mathbf{Y}]_1$ are those that are boolean combinations of languages of the form $\Sigma^* \sigma_1 \sigma_2$, which are exactly the languages whose syntactic monoids are in the variety $\mathbf{D}_2$. Furthermore, the languages definable in $\mathsf{TL}[\mathbf{H}]$ are exactly the languages whose syntactic monoids are in the variety $\mathbf{R}$ (Brzozowski and Fich, 1980). Then, by the semidirect product substitution principle of Thérien and Wilke (2001, Proposition 3.8), all languages definable in $\mathsf{TL}[\mathbf{H}] \circ \mathsf{TL}[\mathbf{Y}]_1$ have syntactic monoids in $\mathbf{R} * \mathbf{D}_2$.

Let M be the syntactic monoid of $(aab)^*$. It suffices to show $M \notin \mathbf{R} * \mathbf{LI}_2$, because $\mathbf{R} * \mathbf{LI}_2 = \mathbf{R} * \mathbf{D}_2$ (Straubing, 1985). We will use Prop. D.1. First, we note that the only surjective morphism $\varphi : \{a, b\}^+ \to M$ is the one mapping elements of $(aab)^*$ to M (up to switching a and

b, but the argument is identical in that case). Secondly, for any $N \in \mathbf{R}$ and any morphism $\psi\colon (\Sigma^2 \times \Sigma \times \Sigma^2) \to N$, we know that for any x, y in the domain it must be the case that $\psi((xy)^\omega) = \psi((xy)^\omega x)$, where ω is the idempotent power of $\psi(xy)$ (Brzozowski and Fich, 1980). Let $x = (aa, b, ab)$ and $y = (ab, a, ba)(ba, a, aa)$. Let $\pi_1 = ((aa, b, ab)(ab, a, ba)(ba, a, aa))^\omega$ and $\pi_2 = ((aa, b, ab)(ab, a, ba)(ba, a, aa))^\omega (aa, b, ab)$. Here, $\mathcal{L}(\pi_1) = \mathcal{L}(\pi_2)$ and $\mathcal{R}(\pi_1) = \mathcal{R}(\pi_2)$ and $\psi(\pi_1) = \psi(\pi_2)$. However $\mathcal{V}(\pi_1) = \varphi(aa(baa)^\omega)$ while $\mathcal{V}(\pi_2) = \varphi(aa(baa)^\omega a)$, which are not equivalent under the syntactic morphism φ. Therefore, by contradiction, $M \notin \mathbf{R} * \mathbf{LI}_2$. □