# Nesting Until and Since in Linear Temporal Logic

Denis Thérien*[1] and Thomas Wilke[2]

[1] School of Computer Science, McGill University, Montréal, Canada
[2] Computer Science Department, CAU, Kiel, Germany

**Abstract.** We provide an effective characterization of the "until-since hierarchy" of linear temporal logic, that is, we show how to compute for a given temporal property the minimal nesting depth in "until" and "since" required to express it. This settles the most prominent classification problem for linear temporal logic. Our characterization of the individual levels of the "until-since hierarchy" is algebraic: for each $n$, we present a decidable class of finite semigroups and show that a temporal property is expressible with nesting depth at most $n$ if and only if the syntactic semigroup of the formal language associated with the property belongs to the class provided. The core of our algebraic characterization is a new description of substitution in linear temporal logic in terms of block products of finite semigroups.

## 1 Introduction

Linear temporal logic is the most fundamental formalism used in computer-aided verification for specifying properties of hard- and software such as integrated circuits and communication protocols [Pnu77]. Its salient feature, which explains part of its success in practice, are temporal operators modeled after natural language constructs. They express basic temporal relations, for instance, "next", "always", "since", and "until". Of all the operators, "since" and "until" are the most powerful and important ones, as they are expressively complete [Kam68] and the only binary operators. Therefore, the most natural way to classify a temporal property and to describe its complexity is by determining the nesting depth in "until" and "since" that is required to express it, that is, by determining its level in the "until-since hierarchy".[1] In a first step towards an effective characterization of this hierarchy, its future-only version (the "until hierarchy") was characterized effectively in [TW96]. In a further step, level 0 of the "until-since hierarchy" was shown to be decidable [TW98]. In the present paper, we finally present effective characterizations of all the other levels of the "until-since hierarchy".

---

* Supported by FCAR, NSERC and Alexander von Humboldt Foundation

[1] Already fairly simple fairness constraints (for instance, "a request is served $k$ times without another request being served") provide properties that show that the "until-since hierarchy" is strict [EW96].

Classifying temporal properties according to their syntactical complexity was first carried out by Sistla and Zuck [SZ87,SZ93] and Cohen-Chesnot [CC89], and has since been a subject of many papers. The objective has been to determine, for a given temporal property, which temporal operators are needed to express it, that is, to characterize fragments of linear temporal logic determined by which of the temporal operators are allowed to be used. Meanwhile, all fundamental fragments have been characterized effectively, by fragments of first-order logics [EVW97,TW98], classes of automata [EW96], or classes of semigroups (monoids) [CPP93].

Technically, we characterize each level of the until-since hierarchy by a corresponding pseudovariety of semigroups. That is, for each level of the hierarchy, we provide a decidable class of finite semigroups and show that a temporal property is definable on the respective level if and only if the transformation semigroup of the minimal automaton recognizing the property defined by the formula viewed as a formal language belongs to the class provided. There are two difficult parts to this characterization: (1) to devise the right classes of semigroups, (2) to show that these classes are decidable. For the second part, we rely on deep results in the theory of finite semigroups and categories [Alm96,Ste99] (Subsect. 4.3). For the first part, we proceed as follows. In a first step (Sect. 3), we establish a connection between substitution in temporal logic and block products of semigroups. This is reminiscent of [TW96], there are, however, fundamental obstacles in the two-sided framework that have to be overcome. Instead of sets of strings (the usual objects of formal language theory), we now study what we call pointed languages, which are sets of pointed strings (strings with a distinguished position). We replace the wreath product by its two-sided version, known as the block product, and prove that if $\Phi$ and $\Psi$ are classes of formulas and $\boldsymbol{V}$ and $\boldsymbol{W}$ are pseudovarieties of semigroups such that the languages expressible in $\Phi$ are the languages recognized by the elements of $\boldsymbol{V}$ and the pointed languages expressible in $\Psi$ are the languages recognized by the elements of $\boldsymbol{W}$, then the block product of $\boldsymbol{V}$ and $\boldsymbol{W}$ recognizes the languages definable by formulas from $\Phi$ with propositions substituted by formulas from $\Psi$—the block product/substitution principle (Theorem 2). We then prove a normal form theorem for each level of the until-since hierarchy, which describes it as an iterated substitution of simple formula classes (Subsect. 4.1), characterize these simple formula classes, and finally use the block product/substitution principle to get a characterization of the levels of the hierarchy (Subsect. 4.2).

This is an extended abstract; many proofs are missing for lack of space. Full details can be found in our techreport attached to this submission.

## 2 Notation and Fundamental Definitions

As usual, $[n] = \{1, \ldots, n\}$ for every natural number $n$.

Strings are sequences of letters over a finite alphabet. The length of a string $x$ is denoted $|x|$. The $i$-th letter of a string $x$ is denoted $x_i$, that is, $x = x_1 \ldots x_{|x|}$. A pointed string over some alphabet $A$ is a pair $(x, i)$ where $x$ is a nonempty string over $A$ and $i \in [|x|]$. A pointed language over some alphabet $A$ is a set of pointed strings over $A$. The set of all pointed strings over $A$ is denoted $A^{\#}$.

### 2.1 Linear Temporal Logic

A *(linear) temporal logic formula (LTL formula)* over some finite set $\Sigma$ of *propositional variables* is built from the boolean constants, $\mathsf{tt}$ and $\mathsf{ff}$, and the elements of $\Sigma$ using boolean connectives and temporal operators: $\oplus$ (next), $\ominus$ (previously), $\boxplus$ (always in the future), $\boxminus$ (always in the past), $\diamondplus$ (eventually in the future), $\diamondminus$ (eventually in the past), $\mathsf{U}$ (until), and $\mathsf{S}$ (since), where only $\mathsf{U}$ and $\mathsf{S}$ are binary and the other operators are unary. For each of the operators except for $\oplus$ and $\ominus$, we also have a nonstrict variant, denoted by $\dot{\boxplus}$, $\dot{\boxminus}$, $\dot{\diamondplus}$, $\dot{\diamondminus}$, $\dot{\mathsf{U}}$, and $\dot{\mathsf{S}}$, respectively.

Given an LTL formula $\varphi$ over some $\Sigma$ and a pointed string $(x,i) \in (2^{\Sigma})^{\#}$, we write $(x,i) \models \varphi$ for the fact that $\varphi$ holds in $x$ at position $i$. This is standard, see, e. g., [Eme90]. The difference between the nonstrict and strict interpretation is as usual, for instance, $(x,i) \models \dot{\diamondplus}\varphi$ if there exists $j$ with $i \leq j \leq |x|$ such that $(x,j) \models \varphi$, whereas $(x,i) \models \diamondplus\varphi$ if there exists $j$ with $i < j \leq |x|$ such that $(x,j) \models \varphi$.

Consider, for instance, the formula $\dot{\diamondplus}(p \wedge q \wedge q \mathbin{\mathsf{U}} (p \wedge q \wedge q \mathbin{\mathsf{U}} p \wedge q))$. This formula expresses that there exists an interval labelled $q$ where at least at three positions $p$ holds true. ($\mathsf{U}$ is assumed to bind stronger than $\wedge$.)

With every formula $\varphi$ over $\Sigma$, we associate a language and a pointed language over $2^{\Sigma}$:

$$L(\varphi) = \{x \in (2^{\Sigma})^{+} \mid (x,1) \models \varphi\} \ , \tag{1}$$

$$P(\varphi) = \{(x,i) \in (2^{\Sigma})^{\#} \mid (x,i) \models \varphi\} \ . \tag{2}$$

We extend these definitions to classes of formulas in the following way. If $\Phi$ is a class of formulas, then

$$L(\Phi) = \{L(\varphi) \mid \varphi \in \Phi\} \ , \qquad P(\Phi) = \{P(\varphi) \mid \in \Phi\} \ . \tag{3}$$

The set of temporal formulas over some set $\Sigma$ of propositional variables is denoted by $\mathsf{LTL}_{\Sigma}$. A function $\sigma \colon \mathsf{LTL}_{\Sigma} \to \mathsf{LTL}_{\Gamma}$ is a *substitution* if $\sigma$ maps each formula $\varphi$ over $\Sigma$ to the formula which is obtained from $\varphi$ by replacing every occurrence of a propositional variable $p \in \Sigma$ by $\sigma(p)$. Note that a substitution is fully determined by the values for the propositional variables. When $\Psi$ is a class of $\mathsf{LTL}_{\Gamma}$ formulas, a substitution $\sigma \colon \mathsf{LTL}_{\Sigma} \to \mathsf{LTL}_{\Gamma}$ is a $\Psi$ substitution if $\sigma(p) \in \Psi$ for every $p \in \Sigma$. When $\Phi$ and $\Psi$ are classes of formulas, then $\Phi \circ \Psi$ is the set of all formulas $\sigma(\varphi)$ where $\varphi \in \Phi$ is a formula over some $\Sigma$ and $\sigma \colon \mathsf{LTL}_{\Sigma} \to \mathsf{LTL}_{\Gamma}$ is a $\Psi$ substitution for some $\Gamma$. Clearly, substitution is associative, so it is not necessary to use parentheses. If $\Phi$ is a class of formula, then $\Phi^0$ denotes the set of all propositional variables and $\Phi^{i+1} = \Phi^i \circ \Phi$.

### 2.2 Until-Since Hierarchy

The *until-since nesting depth* of a temporal formula $\varphi$, denoted $d_{\mathsf{US}}(\varphi)$, is its nesting depth in until and since. Formally, this is defined by

$$
\begin{aligned}
d_{\mathsf{US}}(\mathsf{tt}) = d_{\mathsf{US}}(\mathsf{ff}) = d_{\mathsf{US}}(p) &= 0 \ , &(4)\\
d_{\mathsf{US}}(o_u\varphi) &= d_{\mathsf{US}}(\varphi) \ , &(5)\\
d_{\mathsf{US}}(\varphi\, o_l\, \psi) &= \max(d_{\mathsf{US}}(\varphi), d_{\mathsf{US}}(\psi)) \ , &(6)\\
d_{\mathsf{US}}(\varphi\, o_b\, \psi) &= \max(d_{\mathsf{US}}(\varphi), d_{\mathsf{US}}(\psi)) + 1 \ , &(7)
\end{aligned}
$$

where $p$ is an arbitrary propositional variable, $\varphi$ and $\psi$ are arbitrary formulas, $o_u$ stands for any of the unary operators ($\neg$, $\ominus$, $\oplus$, $\dot{\boxplus}$, $\dot{\boxminus}$, $\dot{\diamondplus}$, $\dot{\diamondminus}$, $\boxplus$, $\boxminus$, $\diamondplus$, $\diamondminus$), $o_l$ stands for $\vee$ or $\wedge$, and $o_b$ stands for any until or since operator ($\dot{\mathsf{U}}$, $\dot{\mathsf{S}}$, $\mathsf{U}$, $\mathsf{S}$).

For every $i$, let $\mathsf{USH}_i$ be the set of all formulas $\varphi$ with $d_{\mathsf{US}}(\varphi) \leq i$, that is, $\mathsf{USH}_i$ is the set of all formulas of until-since depth at most $i$. The *until-since hierarchy* is the following chain:

$$
L(\mathsf{USH}_0) \subseteq L(\mathsf{USH}_1) \subseteq L(\mathsf{USH}_2) \subseteq \dots \tag{8}
$$

The class $L(\mathsf{USH}_i)$ is called the *$i$-th level of the until-since hierarchy.*

In [EW96], it was shown that the until-since hierarchy is strict, that is, each of the containments is proper, and in [TW98], it was shown that $L(\mathsf{USH}_0)$ is decidable.

It should be noted that even though the above definition of the hierarchy might seem to be susceptible to which operators are included in the syntax of temporal logic, this is not at all the case: Even if only $\oplus$, $\ominus$, $\dot{\diamondplus}$, $\dot{\diamondminus}$, $\dot{\mathsf{S}}$, and $\dot{\mathsf{U}}$ were allowed, the hierarchy would be exactly the same. Also, if any of the other binary operators suggested in the literature (Kröger's "at next" [Krö87], Lamport's "at least as long as" [Lam83], or STeP's "wait for" [MAB+94]) were added and taken into account in the above definition of until-since depth ("binary depth"), then the hierarchy would not change either.

### 2.3 Monoids, Semigroups, and Formal Languages

For a complete treatment of the notions presented here, the reader is referred to [Eil76,Alm95].

A semigroup is a set with a binary associative operation; a monoid is a semigroup that contains a two-sided identity element. A monoid $M$ recognizes a language $L \subseteq A^*$ if there is a homomorphism $\theta \colon A^* \to M$ and a subset $F \subseteq M$ such that $L = \theta^{-1}(F)$. Given two monoids $M$ and $N$, we say that $M$ divides $N$ if $M$ is a homomorphic image of a submonoid of $N$. The natural classification for finite monoids is in terms of pseudovarieties, i. e., classes of finite monoids that are closed under division and direct product. If $\boldsymbol{V}$ is a pseudovariety, we write $L(\boldsymbol{V})$ for the class of languages that can be recognized with a monoid in

$\boldsymbol{V}$. The following examples of pseudovarieties are well-known and will be used later:

$$\mathbf{L_1} \vee \mathbf{R_1} = \{M \mid \forall s, t, u \in M(stsus = stus \wedge s = s^2)\} \ , \tag{9}$$

$$\mathbf{DA} = \{M \mid \forall e, s \in M(MeM = MsM \wedge e = e^2 \rightarrow s = s^2)\} \ . \tag{10}$$

Several operations combining two pseudovarieties to yield a third one have been investigated. We here need the so-called *block product*, a 2-sided variant of the more classical wreath product. We will be using the following characterization of this operation.

Let $\gamma$ be a finite-index congruence on $A^*$, and let $N = A^*/\gamma$ be the quotient monoid of $A^*$ with respect to $\gamma$. The $\gamma$-class of a string $x$ will be denoted by $[x]_\gamma$. Let $\beta$ be a finite-index congruence on $(N \times A \times N)^*$. The relation $\beta \,\square\, \gamma$ is the equivalence relation on $A^*$ defined by: $x \ \beta \,\square\, \gamma \ y$ iff $x \ \gamma \ y$ and for all $u, v \in A^*$, $h_{u,v}(x) \ \beta \ h_{u,v}(y)$ where $h_{u,v}(a_1 \ldots a_n) = b_1 \ldots b_n$ with $b_i = ([ua_1 \ldots a_{i-1}]_\gamma, a_i, [a_{i+1} \ldots a_n v]_\gamma)$. It is not hard to verify that $\beta \,\square\, \gamma$ is a congruence of finite index on $A^*$.

**Lemma 1.** **[Thé91]** *Let $\boldsymbol{V}$ and $\boldsymbol{W}$ be pseudovarieties of monoids. A monoid $M = A^*/\alpha$ belongs to the pseudovariety $\boldsymbol{V} \,\square\, \boldsymbol{W}$ iff $\alpha$ is refined by $\beta \,\square\, \gamma$ with $N = A^*/\gamma$ in $\boldsymbol{V}$ and $(N \times A \times N)^*/\beta$ in $\boldsymbol{W}$.*

Note that the block product of pseudovarieties is not associative, i. e., $(\boldsymbol{V} \,\square\, \boldsymbol{W}) \,\square\, \boldsymbol{U}$ is not equal in general to $\boldsymbol{V} \,\square\, (\boldsymbol{W} \,\square\, \boldsymbol{U})$. When writing an iterated block product without parentheses, we mean the first possibility, i. e., bracketing is from left to right.

It turns out that it is sometimes more appropriate to deal with semigroups rather than with monoids, and to consider languages as subsets of $A^+$ rather than $A^*$. All notions defined above have natural correspondence in this slightly different setting. In particular, it is possible to define the block product of $\boldsymbol{V}$ and $\boldsymbol{W}$ when either one is (or both are) a pseudovariety of semigroups rather than monoids. The result is then a pseudovariety of semigroups and a characterization in terms of congruences can be given, in a way very similar to the pure-monoid case. The following pseudovariety of semigroups has special importance in the algebraic theory of automata and it also plays a role in our context:

$$\mathbf{LI} = \{S \mid \forall s, e \in S(e^2 = e \rightarrow ese = e)\} \ . \tag{11}$$

## 3 The Block Product/Substitution Principle

In this section, we present the block product/substitution principle, which establishes a general, fundamental relationship between block products of pseudovarieties of monoids and substitution on classes of formulas. We start with defining what it means for a pointed language to be recognized by a monoid homomorphism or simply a monoid.

Let $h\colon A^* \to M$ be a monoid homomorphism. We define a companion function $\hat{h}$ mapping pointed strings over $A$ to elements of $M \times A \times M$ by

$$\hat{h}(x,i) = (h(x_1 \dots x_{i-1}), x_i, h(x_{i+1} \dots x_{|x|})) \ . \tag{12}$$

We say a pointed language $P$ over $A$ is *recognized* by $h$ if there exists a set $U \subseteq M \times A \times M$ such that $P = \hat{h}^{-1}(U)$. We say that $P$ is recognized by $M$ if it is recognized by a homomorphism $A^* \to M$.

Similar to above, given a class $\boldsymbol{V}$ of finite monoids, we set

$$P(\boldsymbol{V}) = \{P \mid \exists M \in \boldsymbol{V}(P \text{ is recognized by } M)\} \ . \tag{13}$$

In a similar fashion, recognition by semigroups and semigroup homomorphisms can be defined, and $P(\boldsymbol{V})$ can be defined for pseudovarieties of semigroups, too.

The main result on block products and substitution is the following.

**Theorem 2 (block product/substitution principle).** *Let $\Phi$ and $\Psi$ be classes of LTL formulas and $\boldsymbol{V}$ and $\boldsymbol{W}$ pseudovarieties of monoids or semigroups such that $L(\Phi) = L(\boldsymbol{V})$, $L(\Psi) \subseteq L(\boldsymbol{V} \Box \boldsymbol{W})$, and $P(\Psi) = P(\boldsymbol{W})$, then*

$$L(\Phi \circ \Psi) = L(\boldsymbol{V} \Box \boldsymbol{W}) \ . \tag{14}$$

Observe that in this theorem, the assumption that we make for $\Phi$ and $\Psi$ are different: languages recognized vs. pointed languages recognized. This explains the seeming contradiction that we establish a correspondence between associative substitution and non-associate block product.

The two main tools in the proof of this theorem are the substitution lemma (see below) and Lemma 1 given in Section 2.3.

Let $\sigma\colon \mathsf{LTL}_\Sigma \to \mathsf{LTL}_\Gamma$ be a substitution. For each $x \in (2^\Gamma)^+$ of length $n$, let $\sigma^{-1}(x)$ be the string $a_1 \dots a_n$ with

$$a_i = \{p \in \Sigma \mid (x,i) \models \sigma(p)\} \ . \tag{15}$$

**Lemma 3 (LTL substitution lemma).** *Let $\varphi$ be an LTL formula over $\Sigma$ and $\sigma\colon \mathsf{LTL}_\Sigma \to \mathsf{LTL}_\Gamma$ a substitution. Then, for each $i \in [|x|]$,*

$$(x,i) \models \sigma(\varphi) \qquad \textit{iff} \qquad (\sigma^{-1}(x), i) \models \varphi \ . \tag{16}$$

The proof of Theorem 2 roughly goes as follows. Let $\varphi$ be a formula over $\Sigma$ and $\beta$ a congruence on $(2^\Sigma)^*$ such that $L(\varphi)$ is a union of $\beta$-classes. Note that $\beta$ can be chosen such that the corresponding quotient monoid $(2^\Sigma)^*/\beta$ is in $\boldsymbol{V}$. Let $\sigma\colon \mathsf{LTL}_\Sigma \to \mathsf{LTL}_\Gamma$ be a $\Psi$ substitution and $\gamma$ a congruence on $(2^\Gamma)^*$ such that the corresponding quotient monoid $N$ recognizes every pointed language $P(\sigma(p))$ where $p \in \Sigma$. This congruence can be chosen in $\boldsymbol{W}$. Let $\rho\colon (N \times 2^\Gamma \times N)^* \to (2^\Sigma)^*$ be the homomorphism induced by the function mapping $(n_1, a, n_2)$ to $\{p \mid (uav, |u|{+}1) \models \sigma(p) \text{ where } h(u) = n_1, h(v) = n_2\}$; this is well-defined since $N$ recognizes every pointed language of the form $P(\sigma(p))$. Next, let $\hat{\beta}$ be the congruence on $(N \times 2^\Gamma \times N)^*$ defined by $X \ \hat{\beta}\ Y$ iff $\rho(X) \ \beta \ \rho(Y)$. The quotient monoid is clearly in $\boldsymbol{V}$, and one verifies that $L(\sigma(\varphi))$ is a union of $\hat{\beta} \Box \gamma$-classes. The first containment then follows from Lemma 1. For the other containment, we use similar arguments.

# 4 The Until-Since Hierarchy

Recall from Subsection 2.2 that the $i$-th level of the until-since hierarchy comprises all languages that are definable by a formula of nesting depth at most $i$ in until and since. In this section, we describe how we prove that for each $i$ it is decidable whether or not a given regular language (say given by a regular expression or a finite automaton) belongs to the $i$-th level. Our decision procedure is uniform in the sense that given $L$, the minimal level it belongs to and an equivalent formula of minimal nesting depth can be computed.

## 4.1 Iterated Substitution

To be able to apply the block product/substitution principle, we describe $\mathsf{USH}_i$ as an iterated substitution of simple formula classes.

It is convenient to introduce two more unary temporal operators, denoted ¢ and \$. Their semantics is given by $(x,i) \models ¢\varphi$ if $(x,1) \models \varphi$, and $(x,i) \models \$\varphi$ if $(x,|x|) \models \varphi$, that is, they allow to jump from any position to the beginning and the end of a string. (Observe that $¢\varphi$ is equivalent with $\dot{\Diamond}^{-}(\boxminus \mathsf{ff} \wedge \varphi)$; a symmetric statement holds for $\$\varphi$.)

Let $\mathsf{TL}_\Box$ be the class of all LTL formulas where $\dot{\Diamond}^{+}$, $\dot{\Diamond}^{-}$, $\dot{\boxplus}$, and $\dot{\boxminus}$ are the only temporal operators allowed. Further, let $\mathsf{TL}_\bigcirc$ be the class of all LTL formulas where $\oplus$, $\ominus$, ¢, and \$ are the only temporal operators allowed. Finally, a formula belongs to $\mathsf{TL}_{\mathsf{US}}$ if it is a boolean combination of propositional variables, formulas of the form $\varphi \mathbin{\mathsf{U}} \psi$, $\varphi \mathbin{\mathsf{S}} \psi$, $¢(\varphi \mathbin{\dot{\mathsf{U}}} \psi)$, and $\$(\varphi \mathbin{\dot{\mathsf{S}}} \psi)$, where $\varphi$ and $\psi$ are propositional formulas.

**Proposition 4 (normal form theorem).** *For every $i$,*

$$L(\mathsf{USH}_i) = L(\mathsf{TL}_\Box \circ \mathsf{TL}^i_{\mathsf{US}} \circ \mathsf{TL}_\bigcirc) \ . \tag{17}$$

*(Recall that $\mathsf{TL}^i_{\mathsf{US}}$ is an iterated substitution, bracketed from left to right, see Subsection 2.1.)*

This proposition is based on the fact that $\oplus$ and $\ominus$ can always be pushed inside, which is folklore (see, e. g., [EW96]), and the following switching rules which show that $\dot{\Diamond}^{-}$ and $\dot{\boxminus}$ can be "pulled out".

**Lemma 5.** *For all LTL formulas $\varphi$, $\psi$, $\chi$, the following is true:*

$$(\varphi \vee \dot{\Diamond}^{-}\psi) \mathbin{\dot{\mathsf{U}}} \chi \equiv \varphi \mathbin{\dot{\mathsf{U}}} \chi \vee (\dot{\Diamond}^{-}\psi \wedge \dot{\Diamond}^{+}\chi) \vee (\varphi \mathbin{\dot{\mathsf{U}}} \psi \wedge \dot{\Diamond}^{+}\chi) \ , \tag{18}$$

$$(\varphi \vee \dot{\boxminus}\psi) \mathbin{\dot{\mathsf{U}}} \chi \equiv \varphi \mathbin{\dot{\mathsf{U}}} \chi \vee \dot{\Diamond}^{+}(\chi \wedge \dot{\boxminus}(\chi \vee \dot{\boxminus}\psi)) \ , \tag{19}$$

$$\begin{aligned} \varphi \mathbin{\dot{\mathsf{U}}} (\psi \wedge \dot{\Diamond}^{-}\chi) \equiv{}& (\dot{\Diamond}^{-}\chi \wedge \varphi \mathbin{\dot{\mathsf{U}}} \psi) \vee (\dot{\boxplus}\varphi \wedge \dot{\Diamond}^{+}(\chi \wedge \dot{\Diamond}^{+}\psi)) \\ & \vee (\neg\dot{\Diamond}^{-}\chi \wedge \dot{\Diamond}^{+}\neg\varphi \wedge \dot{\boxplus}(\varphi \vee \dot{\Diamond}^{-}(\psi \wedge \dot{\Diamond}^{-}\chi))) \ , \end{aligned} \tag{20}$$

$$\varphi \mathbin{\dot{\mathsf{U}}} (\psi \wedge \dot{\boxminus}\chi) \equiv \dot{\boxminus}\chi \wedge ((\varphi \wedge \chi) \mathbin{\dot{\mathsf{U}}} \psi) \ . \tag{21}$$

We also need future-only counter-parts of these switching rules, but those were already proved in [TW96].

### 4.2 Algebraic Characterizations

In this section we provide algebraic classifications of the small classes of formulas we have seen above ($\mathsf{TL}_{\bigcirc}$, $\mathsf{TL}_{\Box}$, and $\mathsf{TL}_{\mathsf{US}}$) and each level of the until-since hierarchy.

For $\mathsf{TL}_{\Box}$, a characterization is known:

**Theorem 6.** **[TW98]** *$L(\mathsf{TL}_{\Box}) = L(\mathbf{DA})$.*

For $\mathsf{TL}_{\mathsf{US}}$ and $\mathsf{TL}_{\bigcirc}$, we prove:

**Lemma 7.** *$P(\mathsf{TL}_{\mathsf{US}}) = P(\mathbf{L_1} \vee \mathbf{R_1})$ and $P(\mathsf{TL}_{\bigcirc}) = P(\mathbf{LI})$.*

As a consequence of the previous lemma, Theorem 6, Proposition 4, and the block product/substitution principle, we obtain:

**Theorem 8 (algebraic characterization of the until-since hierarchy).** *For every $i$,*

$$L(\mathsf{USH}_i) = L(\mathbf{DA} \,\Box\, (\mathbf{L_1} \vee \mathbf{R_1})^i \,\Box\, \mathbf{LI}) \ . \tag{22}$$

### 4.3 Decidability

In this section we will describe how to decide if a given language $L$ belongs to the $i$-th level of the until-since hierarchy, that is, to $L(\mathsf{USH}_i)$. By Theorem 8, this is equivalent to deciding if a semigroup belongs to the semigroup variety $\boldsymbol{V}_i \,\Box\, \mathbf{LI}$ where $\boldsymbol{V}_i$ is the monoid variety given by

$$\boldsymbol{V}_0 = \mathbf{DA} \ , \tag{23}$$

$$\boldsymbol{V}_i = \boldsymbol{V}_{i-1} \,\Box\, (\mathbf{L_1} \vee \mathbf{R_1}) \ . \tag{24}$$

The algorithms for solving these problems are based on deep semigroup-theoretic results, of which we present only the main points.

It is now well understood that the study of decompositions based on block (or wreath) products involves the framework of categories, monoids being then viewed as special, one-object, categories. Several notions, such as division, carry over to the more general setting in a natural way. See, for instance, [Til87]. The following lemma is crucial.

**Lemma 9.** *It is decidable if a given category divides a monoid in $\boldsymbol{V}_i$.*

This is proved by induction on $i$. The inductive base follows directly from a result in [Alm96]; the inductive step relies heavily on Theorem 9.2 of [Ste99].

The next lemma we use is:

**Lemma 10.** *For any nontrivial monoid variety $\boldsymbol{V}$,*

$$\boldsymbol{V} \,\Box\, \mathbf{LI} = \boldsymbol{V} * \mathbf{D} \ , \tag{25}$$

*where $*$ denotes the semidirect product on pseudovarieties of semigroups.*

So, as a consequence, we know that $\boldsymbol{V}_i \,\square\, \mathbf{LI} = \boldsymbol{V}_i * \mathbf{D}$, and it is enough to show decidability of the latter. By [Str85], as interpreted in the language of categories in [Til87], to decide membership of $S$ in $\boldsymbol{V} * \mathbf{D}$ it suffices to show that a certain category, which is explicitly constructible from $S$, divides a monoid in $\boldsymbol{V}$. This is possible for $\boldsymbol{V}_i$ by Lemma 9. We conclude:

**Theorem 11 (decidability of the until-since hierarchy).** *For every $i$, the $i$-th level of the until hierarchy is decidable, that is, given a regular language $L$ it can be determined whether $L \in L(\mathsf{USH}_i)$.*

## 5 Conclusion

With the block product/substitution principle we have provided a very powerful tool for characterizing fragments of linear temporal logic. Using this and deep results from finite semigroup theory, we have been able to provide effective characterization of all levels of the until-since hierarchy, the most natural hierarchy for classifying temporal properties. We know how to extend Theorem 11 to $\omega$-languages for $i = 0$ and $i = 1$, and it is possible that the technique we apply in these cases can be generalized to higher levels.

*Acknowledgment.* We thank the anonymous referees for their helpful comments.

## References


[Alm95] Jorge Almeida. *Finite Semigroups and Universal Algebra*, volume 3 of *Series in Algebra*. World Scientific, Singapore, 1995.

[Alm96] Jorge Almeida. A syntactical proof of locality of **DA**. *Internat. J. Algebra and Comput.*, 6(2):165–177, 1996.

[CC89] Joëlle Cohen-Chesnot. *Etude algébrique de la logique temporelle*. Doctoral thesis, Université Paris 6, Paris, France, April 1989.

[CPP93] Joëlle Cohen, Dominique Perrin, and Jean-Eric Pin. On the expressive power of temporal logic. *J. Comput. System Sci.*, 46(3):271–294, June 1993.

[Eil76] Samuel Eilenberg. *Automata, Languages, and Machines*, volume 59-B of *Pure and Applied Mathematics*. Academic Press, New York, 1976.

[Eme90] Allen E. Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Methods and Semantics, pages 995–1072. Elsevier Science Publishers B. V., Amsterdam, 1990.

[EVW97] Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. In *Proceedings 12th Annual IEEE Symposium on Logic in Computer Science*, pages 228–235, Warsaw, Poland, 1997.

[EW96] Kousha Etessami and Thomas Wilke. An until hierarchy for temporal logic. In *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*, pages 108–117, New Brunswick, N. J., 1996.

[Kam68] Johan Anthony Willem Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, Calif., 1968.

[Krö87] Fred Kröger. *Temporal Logic of Programs.* Springer, 1987.

[Lam83] Leslie Lamport. Specifying concurrent program modules. *ACM Trans. Programming Lang. Sys.*, 5(2):190–222, 1983.

[MAB+94] Zohar Manna, Anuchit Anuchitanukul, Nikolaj Bjørner, Anca Browne, Edward Chang, Michael Colón, Luca de Alfaro, Harish Devarajan, Henny Sipma, and Tomas Uribe. STeP: the Stanford Temporal Prover. Technical Report STAN-CS-TR-94-1518, Dept. of Computer Science, Stanford University, Stanford, Calif., 1994.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57, Rhode Island, Providence, 1977.

[Ste99] Ben Steinberg. Semidirect products of categories and applications. *Journal of Pure and Applied Algebra*, 142:153–182, 1999.

[Str85] Howard Straubing. Finite semigroup varieties of the form $\boldsymbol{V} * \mathbf{D}$. *J. Pure Appl. Algebra*, 36:53–94, 1985.

[SZ87] A. Prasad Sistla and Lenore D. Zuck. On the eventuality operator in temporal logic. In *Proceedings, Symposium on Logic in Computer Science*, pages 153–166, Ithaca, New York, June 1987.

[SZ93] A. Prasad Sistla and Lenore D. Zuck. Reasoning in a restricted temporal logic. *Inform. and Computation*, 102(2):167–195, February 1993.

[Thé91] Denis Thérien. Two-sided wreath product of categories. *Journal of Pure and Applied Algebra*, 74:307–315, 1991.

[TW96] Denis Thérien and Thomas Wilke. Temporal logic and semidirect products: An effective characterization of the until hierarchy. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 256–263, Burlington, Vermont, 1996.

[TW98] Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as one quantifier alternation: $\mathrm{FO}^2 = \Sigma_2 \cap \Pi_2$. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 41–47, Dallas, Texas, May 1998.

[Til87] Brat Tilson. Categories as algebra. *J. Pure Appl. Algebra*, 48:83–198, 1987.