Characterizing EF and EX tree logics

Mikolaj Bojańczyk and Igor Walukiewicz
Warsaw University and
Laboratoire Bordelais de Recherche en Informatique

Abstract

We describe the expressive power of temporal branching time logics that use the modalities EX and EF. We give a forbidden pattern characterization of the tree languages definable in three logics: EX, EF and EX+EF. The properties in these characterizations can be verified in polynomial time when given a minimal deterministic bottom-up tree automaton.

We consider the definability problem for logics over binary trees: given a regular tree language decide if it can be expressed by a formula of the logic in question. The main motivation for considering this problem is to understand the expressive power of tree logics. Although a very old question, definability has gained new relevance with the XML community's burgeoning interest in tree models [8]. Indeed, numerous new formalisms for describing tree properties have been recently proposed.

For words the definability question is well studied and understood. Starting from the celebrated Schutzenberger theorem [12], characterizing star-free word languages by aperiodicity, numerous other language classes have been classified. In particular, we now have a good understanding of the expressive power of LTL and its fragments [14, 18]. This is in sharp contrast with the case of trees where much less is known.

We feel that the major goal in the study of the definability problem for trees is to characterize the expressive power of first-order logic, or equivalently CTL*[1] (we consider finite binary trees here). It seems however that this is a difficult problem whose solution demands new tools and expertise. This is why we have decided to consider fragments of CTL* where the problem is easier. The fragments in question use the modalities EX (there is a successor) and EF (there is a descendant). Apart from being a step towards solving the first-order definability problem, these fragments are interesting on their own. The model-checking problem for them is easier than for CTL*, and even than for CTL: for example when a model is given by a BPP [2] or by a a push-down system [16]. The modalities EX and EF are also closely related to operators in XPath [5, 4].

We prove the definability problem decidable for three logics: EX, EF and EX+EF. These are built by using the eponymous modalities along with boolean connectives. Our decision procedures use a sort of forbidden pattern characterizations. These forbidden patterns are expressed in terms of the minimal leaves-to-root automaton recognizing a given tree language. The resulting algorithms are polynomial in the number of states of the minimal automaton. If, on the other hand, we assume that the input is a CTL formula or a nondeterministic tree automaton then we obtain the EXPTIME upper bound matching the obvious lower bound for the problem.

As mentioned above not much is known about the definability problem for trees. There exist basic results: characterizations of the class of regular tree languages by monadic second-order logic [15] or the μ-calculus [9]; equivalence of first-order logic and CTL* over finite binary trees [6]. Yet there is no equivalent of the Schutzenberger theorem for trees; in fact decidability of first order definability is still open. There has been some work in this direction; in particular borrowing the notion of aperiodicity from the word case is known to be insufficient [11, 7].

It is also a valid question to compare the characterizations presented in this paper with the ones in [18] for the corresponding word logics X, F and $\mathsf{X} + \mathsf{F}$. Although there is some resemblance between the two, our results need more than a straightforward extension of the forbidden pattern characterizations from the word case. This is in a way unfortunate because it suggests that an equivalent of the Schutzenberger theorem for trees may also require an intricate extension of the aperiodicity.

The plan of the paper is as follows. After a preliminary section we briefly state a characterization of EX logic. This is very similar to a characterization of modal logics presented in the literature [10], so we mention the result mostly for completeness. In the next two sections we respectively characterize the logics EF and EX+EF. Maybe counterintuitively, the argument for the weaker EF logic is longer. In the penultimate section we summarize the results, showing how they imply decidability algorithms. Finally, we justify our characterizations by pointing out why the forbidden patterns known from the word case do not adapt directly to the tree case.

A *tree domain* is any nonempty finite prefix-closed subset of $\{0, 1\}^*$. A Σ-*tree* is any mapping from some tree domain to Σ. We denote by $\mathrm{dom}(t)$ the domain of t (as a function), elements of this set are called *nodes*. The *root* is another name for the node ϵ. Nodes in a tree are ordered by the prefix relation, which is denoted $\leq$. *Leaves* are nodes that are maximal with respect $\leq$. The *subtree* of a tree t rooted in the node v is the tree that assigns to a node w the label $t(v \cdot w)$. This tree is denoted as $t|_v$ and its domain is the set of nodes w such that $v \cdot w$ is a node of t. The left subtree of t is the tree $t|_0$, the right subtree is $t|_1$. A Σ-*context* is any Σ tree with a distinguished leaf, which is called the *hole*. We denote contexts by $C[]$, $D[]$. We write $C[t]$ to denote the tree obtained from the context $C[]$ by substituting the tree t for the hole.

A Σ-language is any set of Σ-trees. Given a language L, two trees s and t are said to be L-*equivalent* if for every context $C[]$, either both or none of the trees $C[s]$ and $C[t]$ belong to L. This is an equivalence relation. An equivalence class of this relation is called an L-*type*. We will denote types using the letters α, β, γ. Given two L-types α, β and a letter a, the L-type $a[\alpha, \beta]$ is uniquely defined and contains the trees equivalent to any tree with a in the root, s in the left son and t in the right son. A language is *regular* if it has a finite number of types.

1 EX+EF Formulas

EX+EF formulas are CTL* formulas which use boolean connectives, letter symbols and where the temporal modalities are allowed only in the forms EX (exists next) and EF (exists finally).

Below follows a more formal definition along with the semantics. Let Σ be an alphabet. The set of EF+EX formulas over Σ is the smallest set of formulas such that:

- Every letter $a \in \Sigma$ is a formula.
- Boolean combinations $\neg\psi$, $\psi \wedge \varphi$ and $\psi \vee \varphi$ of formulas are also formulas.
- If φ is a formula then $\mathsf{EX}_0\varphi$, $\mathsf{EX}_1\varphi$ and $\mathsf{EF}\varphi$ are formulas.

The set of EF formulas is the fragment obtained by disallowing EX_0 and EX_1, while the set of EX formulas is obtained by disallowing EF. We now define the semantics, which with every formula associates a set of Σ-trees that *satisfy* it:

- A tree satisfies the formula a if its root is labeled by a;
- Satisfaction for boolean operations is defined in the standard way;
- A tree satisfies $\mathsf{EX}_0\varphi$ (resp. $\mathsf{EX}_1\varphi$) if its left (resp. right) subtree satisfies φ;

- A tree satisfies $\mathsf{EF}\varphi$ if it has a proper subtree that satisfies φ.

We call EF, EX_0 and EX_1 the *modalities*. Observe that the modality EF has strict semantics here: the appropriate subtree has to be proper. The formula $\mathsf{AX}\varphi$ is used as an abbreviation of $\neg\mathsf{EX}\neg\varphi$, while $\mathsf{AG}\varphi$ is used as an abbreviation of $\neg\mathsf{EF}\neg\varphi$.

A tree language L is definable in EF+EX if there is a formula satisfied in exactly the trees in L. Similarly we define languages definable in EF and in EX.

2 Languages Definable in EX

In this section we state a characterization of languages definable in EX. We do this for the sake of completeness since the characterization is essentially the same as in [10].

Definition 1 Two trees are *identical up to depth k* if they are the same when restricted to $\{0,1\}^{\leq k}$. We say that a language L is *dependent on depth k* if every two trees which are identical up to depth k have the same L-type.

A context is *nontrivial* if its hole is not in the root.

Definition 2 Let L be a language and let α, β be two distinct L-types. We say that the language L contains an *$\{\alpha,\beta\}$-loop* if for some nontrivial context $C[]$, both $C[\alpha] = \alpha$ and $C[\beta] = \beta$ hold.

Theorem 3
For a regular language L, the following conditions are equivalent:

1. *L is definable in EX;*
2. *For some $k \in \mathcal{N}$, L is dependent on depth k;*
3. *L does not have an $\{\alpha,\beta\}$-loop for any two L-types α, β .*

Proof
The equivalence of the first two conditions is obvious, as is the implication from 2 to 3. To end the proof of the theorem, we will show that if the language L is not dependent on any depth k, then a loop can be found.

Let $k > |\mathrm{Types}(L)|^2$ and assume that L is not dependent on depth k. This means there are trees s and t which are identical up to depth k but have different types. Let $v_1, \ldots, v_n$ be all the nodes of depth k in the tree s (or equivalently in t). We define a sequence of trees $s = s_0, \ldots, s_n = t$ which gradually morphs from the tree s to the tree t:

$$s_0 = s \qquad \text{and} \qquad s_i = s_{i-1}[v_i := t|_{v_i}] \text{ for } i > 0 \ .$$

Since the trees s_0 and s_n have different types, there must be some $i \in \{1, \ldots, n\}$ such that s_{i-1} and s_i have different types. These two latter trees differ only below the node v_i. Let $w_0 < \ldots < w_{k-1}$ be all the ancestors of the node v_i. Given $j < k$, let

$$\alpha_j = \mathit{type}(s_{i-1}|_{w_j}) \qquad \text{and} \qquad \beta_j = \mathit{type}(s_i|_{w_j}).$$

Since the node v_i is at depth $k > |\mathrm{Types}(L)|^2$, there must be some two indices $j < k$ such that the equalities $\alpha_j = \alpha_k$ and $\beta_j = \beta_k$ hold. Since the types of s_{i-1} and s_i are distinct, so are the types α_j and β_j. But this means that the part of s_{i-1} whose root is in w_j and whose hole is in w_k provides an $\{\alpha_j, \beta_j\}$-loop. $\square$

3 Languages definable in EF

In this section we show a characterization of languages definable in EF. This is the most involved section of the paper, with a long technical proof.

The characterization result, Theorem 5, shows that definability in EF is equivalent to a certain (decidable) property of the language's types. This property, however, is not directly stated in terms of types, but using an intermediate concept called the delayed type. The intuition behind a delayed type is that it is supposed to contain all the information about a tree that does not depend on the root label.

Given a Σ-tree t and a letter $a \in \Sigma$, we write $t\langle a\rangle$ to denote the tree obtained from t by relabeling the root with the letter a. With every Σ-tree t we associate its *delayed type*, which is the function that assigns to a letter $a \in \Sigma$ the L-type of the tree $t\langle a\rangle$. We will denote delayed types using the letters x, y, z. Note that the delayed type of a tree is uniquely determined by the types of its left and right subtrees. We write $(x, a) \trianglelefteq_L y$ if there is a tree of delayed type y having a subtree of type $x(a)$. We also write $x \trianglelefteq_L y$ if $(x, a) \trianglelefteq_L y$ for some $a \in \Sigma$. This relation is a quasiorder but not necessarily a partial order, since it need not be antisymmetric.

For delayed types x, y and letters $a, b \in \Sigma$, we write $\mathit{dtype}_L(x, a, y, b)$ for the delayed type which assigns to a letter c the type $c[x(a), y(b)]$. In other words, this is the delayed type of a tree whose left and right subtrees have types $x(a)$ and $y(b)$ respectively. The set of *neutral letters* of a delayed type x is the set

$$N_x^L = \{a : x = dtype_L(x, a, x, a)\}.$$

This set may be empty.

Definition 4 A Σ-language L is *EF-admissible* if it is a regular tree language such that all delayed types x, y and all letters $a, c \in \Sigma$ satisfy:

P1 The relation $\trianglelefteq_L$ on delayed types is a partial order;

P2 $dtype_L(x, a, y, b) = dtype_L(x, a, y, b')$ for all $b, b' \in N_y^L$;

P3 if $(x, a) \trianglelefteq_L y$ then $dtype_L(x, a, y, c) = dtype_L(y, c, y, c)$;

P4 $dtype_L(x, a, y, c) = dtype_L(y, c, x, a)$.

Another important concept used in Theorem 5 is that of typeset dependency. The *typeset* of a tree is the set of types of its proper subtrees. We say that a regular language is *typeset dependent* if the delayed type of a tree depends only on its typeset.

Our characterization of EF is presented in the following theorem:

Theorem 5
For every language L, the following conditions are equivalent:

1. *L is definable in EF,*
2. *L is typeset dependent,*
3. *L is EF-admissible.*

The proof of this theorem is long and will be spread across the next three sections; the implications $1 \Rightarrow 2$, $2 \Rightarrow 3$ and $3 \Rightarrow 1$ being proved in Sections 3.1, 3.2 and 3.3 respectively. For the remainder of Section 3 we assume that an alphabet Σ along with a regular Σ-language L are fixed, hence we will omit the L qualifier from the notation, writing for instance $\trianglelefteq$ instead of $\trianglelefteq_L$. We may assume regularity since all conditions 1, 2 and 3 imply this.

3.1 A Language Definable in EF Is Typeset Dependent

In this section, we will show that the language L is typeset dependent using the assumption that it is defined by some EF formula ψ.

Definition 6 By $cl(\psi)$ we denote the smallest set of formulas that contains ψ and is closed under negations and subformulas.

It is not difficult to see that the type of a tree is determined by the set of those formulas from $cl(\psi)$ which it satisfies (although this correspondence need not be injective). Our first step is to show that for the delayed type, even less information is sufficient

Definition 7 An *existential formula* is a formula of the form $\mathsf{EF}\varphi$. The *signature* $\mathrm{Sig}(t)$ of a tree t is the set of existential formulas from $cl(\psi)$ that it satisfies.

Lemma 8 The signature of a tree determines its delayed type.

Proof
Take two trees s and t with the same signatures. For a given letter $a \in \Sigma$, an easy induction on formula size shows that for all $\varphi \in cl(\psi)$:

$$s\langle a\rangle \models \varphi \qquad \text{iff} \qquad t\langle a\rangle \models \varphi.$$

This is due to the fact that the modality EX is strict. Since the two trees $s\langle a\rangle$ and $t\langle a\rangle$ satisfy the same formulas from $cl(\psi)$, their types must be the same. As the choice of the letter a was arbitrary, this implies that the trees s and t have the same delayed types. □

Given two trees t_0, t_1 and a letter $a \in \Sigma$, we write $\mathrm{Sig}(t_0, t_1)$ instead of $\mathrm{Sig}(a[t_0, t_1])$. This notation is unambiguous since $\mathrm{Sig}(a[t_0, t_1])$ does not depend on the letter a.

Given two types α and β, we denote by $dtype(\alpha, \beta)$ the delayed type which assigns to a letter a the type $a[\alpha, \beta]$. A type α is *reachable* from a type β, denoted $\beta \preccurlyeq \alpha$, if $C[\beta] = \alpha$ holds for some context $C[]$. This relation is a quasiorder and we use $\approx$ for the accompanying equivalence relation. The following simple lemma is given without a proof:

Lemma 9 If t' is a subtree of t, then $\mathrm{Sig}(t', s) \subseteq \mathrm{Sig}(t, s)$. If $\alpha \preccurlyeq \beta$ then $dtype(\alpha, \beta) = dtype(\beta, \beta)$.

The following lemma shows that for languages definable in EF, the relation $\approx$ is a congruence with respect to the function $dtype(\alpha, \beta)$:

Lemma 10 If $\alpha_0 \approx \beta_0$ and $\alpha_1 \approx \beta_1$ then $dtype(\alpha_0, \alpha_1) = dtype(\beta_0, \beta_1)$.

Proof
Since a language definable in EF satisfies $dtype(\alpha, \beta) = dtype(\beta, \alpha)$, it is sufficient to prove the case where $\beta_1 = \alpha_1$. Let $C[]$ be a context such that $C[\alpha_0] = \beta_0$ and let $D[]$ be a context such that $D[\beta_0] = \alpha_0$. Both contexts exist by assumption that $\alpha_0 \approx \beta_0$. Let s_0 be a tree of type α_0 and let s_1 be a tree of type α_1. Consider the two sequences of trees $\{s_0^i\}_{i \geq 0}$ and $\{t_0^i\}_{i \geq 0}$ defined by induction as follows:

$$\begin{array}{rcll} s_0^0 & = & s_0; & \\ t_0^i & = & C[s_0^i] & \text{for } i \geq 0; \\ s_0^i & = & D[t_0^{i-1}] & \text{for } i \geq 1. \end{array}$$

By a simple induction one can prove that for all $i \geq 0$,

$$type(s_0^i) = \alpha_0 \quad \text{and} \quad type(t_0^i) = \beta_0 \ .$$

From Lemma 9 we obtain the following inclusions:

$$\mathrm{Sig}(s_0^0, s_1) \subseteq \mathrm{Sig}(t_0^0, s_1) \subseteq \mathrm{Sig}(s_0^1, s_1) \subseteq \mathrm{Sig}(t_0^1, s_1) \subseteq \ \ldots$$

Since there are only finitely many signatures, there must be some $i > 0$ such that $\mathrm{Sig}(s_0^i, s_1) = \mathrm{Sig}(t_0^i, s_1)$. Consequently, by Lemma 8, the delayed types $dtype(\alpha_0, \alpha_1)$ and $dtype(\beta_0, \alpha_1)$ are equal. □

We are now ready to show that the language L is typeset dependent. Let s and t be two trees with the same typeset. We want are going to show that they have the same delayed type.

If this typeset is empty, then both trees have one node and, consequently, the same delayed type. Otherwise one can consider the following four types, which describe the left and right subtrees of s and t:

$$\alpha_0 = type(s|_0) \qquad \alpha_1 = type(s|_1) \qquad \beta_0 = type(t|_0) \qquad \beta_1 = type(t|_1).$$

We need to prove that $dtype(\beta_0, \beta_1) = dtype(\alpha_0, \alpha_1)$. By assumption that the typesets of s and t are equal, both β_0 and β_1 occur in nonroot nodes of s and both α_0 and α_1 occur in nonroot nodes of t. Thus $\beta_0 \preccurlyeq \alpha$ holds for some $\alpha \in \{\alpha_0, \alpha_1\}$ and similarly for β_1, α_0 and α_1. The result follows from the following case analysis:

- $\beta_0, \beta_1 \preccurlyeq \alpha$ for some $\alpha \in \{\alpha_0, \alpha_1\}$. By assumption we must have $\alpha \preccurlyeq \beta$ for some $\beta \in \{\beta_0, \beta_1\}$. Hence $\alpha \approx \beta$. By Lemma 10 we get

$$dtype(\alpha, \alpha) = dtype(\beta, \beta) \ . \tag{1}$$

As $\beta_0, \beta_1 \preccurlyeq \alpha \preccurlyeq \beta$, from Lemma 9 we obtain

$$dtype(\beta_0, \beta_1) = dtype(\beta, \beta) \ . \tag{2}$$

Similarly one proves the equality

$$dtype(\alpha_1, \alpha_2) = dtype(\alpha, \alpha) \ . \tag{3}$$

The three equations (1), (2) and (3) yield the desired result.

- $\alpha_0, \alpha_1 \preccurlyeq \beta$ for some $\beta \in \{\beta_0, \beta_1\}$. As in the case above.
- A short analysis reveals that if neither of the above holds then $\beta_0 \preccurlyeq \alpha_i \preccurlyeq \beta_0$ and $\beta_1 \preccurlyeq \alpha_{1-i} \preccurlyeq \beta_1$ for some $i \in \{0, 1\}$. Therefore $\beta_0 \approx \alpha_i$ and $\beta_1 \approx \alpha_{1-i}$ and an application of Lemma 10 yields the desired result.

3.2 A Typeset Dependent Language Is EF-Admissible

This step of the proof consists of verifying that all the properties P1 to P4 are satisfied if the language is typeset dependent.

Lemma 11 *L satisfies the property P1.*

Proof
Condition P1 states that $\trianglelefteq$ is a partial order on delayed types. The relation $\trianglelefteq$ is obviously transitive and reflexive. We will show that $x \trianglelefteq y \trianglelefteq x$ implies that the delayed types x and y are equal. Assume then that $x \trianglelefteq y \trianglelefteq x$. In this case, for arbitrary n we can find a tree t with nodes $v_1 \leq w_1 \leq v_2 \leq w_2 \leq \cdots \leq v_n$ such that subtrees rooted in the v_i nodes have type x and subtrees rooted in the w_i nodes have type y. Clearly for all $0 \leq i < n$ we have

$$\mathrm{TS}(t|_{v_i}) \subseteq \mathrm{TS}(t|_{w_i}) \subseteq \mathrm{TS}(t|_{v_{i+1}}).$$

If we take n to be bigger than the number of types in L then we can find some i such that $\mathrm{TS}(t|_{v_i}) = \mathrm{TS}(t|_{w_i})$, which by typeset dependency implies $x = y$. $\square$

Lemma 12 *L satisfies the property P2.*

Proof
Condition P2 states that if b, b' are neutral letters for a delayed type y, then the delayed types $dtype(x, a, y, b)$ and $dtype(x, a, y, b')$ are equal. To show

that this condition is satisfied, we define by induction a sequence of trees $t_0, t_1, \ldots$ in the following manner. For t_0 we take some tree of delayed type y with b in the root, while t_{i+1} is defined as $b[b'[t_i, t_i], b'[t_i, t_i]]$, see Fig. 1. Because b and b' are neutral letters for the delayed type y, all the trees t_i have delayed type y. Moreover, for some $j > 0$, the typesets of the trees t_j and $b'[t_j, t_j]$ are equal.

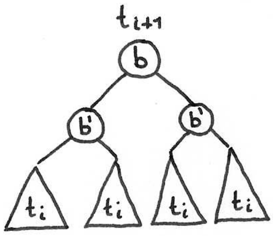

Figure 1: The tree t_{i+1}.

Take now some tree t of delayed type x and with the label a in the root. Let s be a tree with t as the left subtree and t_j as the right subtree. Similarly we define s', but with $b'[t_j, t_j]$ in the right subtree. We do not specify the root letters, since we are interested in delayed types. By assumption on t_j and $b'[t_j, t_j]$, the trees s and s' have the same typesets. Since L is typeset dependent, their delayed types must be equal. Therefore,

$$dtype(x, a, y, b) = dtype(s) = dtype(s') = dtype(x, a, y, b').$$

$\square$

The last two properties P3 and P4 are obviously satisfied in every typeset dependent language.

3.3 A EF-Admissible Language Is Definable EF

We now proceed to the most difficult part of the proof, where a defining EF formula is found based only on the assumption that the properties P1 to P4 are satisfied. We start by stating a key property of EF-admissible languages which shows the importance of neutral letters.

Lemma 13 If the delayed type of a tree t is y, then its every proper subtree with delayed type y has its root label in N_y.

Proof
Consider some proper subtree $t|_v$ of delayed type y and its root label $b =$

$t(v)$. Let w be the brother of the node v and let z, c be its delayed type and label, respectively. Obviously $(z,c) \trianglelefteq y$. By property P3 we get $\mathit{dtype}(y,b,z,c) = \mathit{dtype}(y,b,y,b)$ and consequently $\mathit{dtype}(y,b,y,b) \trianglelefteq y$. As $\trianglelefteq$ is a partial order by P1 and since $y \trianglelefteq \mathit{dtype}(y,b,y,b)$ holds by definition, we get $\mathit{dtype}(y,b,y,b) = y$. Hence b belongs to set N_y of neutral letters of y. $\Box$

Note that if the trees t and $t|_v$ have delayed type y, then so does the tree $t|_w$ for any $w < v$, because $\trianglelefteq$ is a partial order. In particular, the above lemma says that nodes with delayed type y form cones whose non-root elements have labels in N_y.

Formulas Defining Delayed Types

A delayed type x is *definable* if there is some EF formula θ_x true in exactly the trees that have delayed type x. The construction of the θ_x formulas will proceed by induction on the $\trianglelefteq$ order.

A set A of delayed types is *downward closed* if it contains every delayed type $\trianglelefteq$-smaller than an element of A. The first step is the following lemma:

Lemma 14 Let x be a delayed type and let $A \not\ni x$ be a downward closed set of definable delayed types. There is a EF formula fork_x^A such that:

$$t \vDash \mathit{fork}_x^A \quad \text{iff} \quad \mathit{dtype}(t) = x \text{ and for all } w > \epsilon,\ \mathit{dtype}(t|_w) \in A.$$

A *fork* formula is satisfied in a maximal node of delayed type x, where moreover all descendants have delayed types in A.

We postpone the technical proof of this Lemma until Section 3.4. Meanwhile, we will use this lemma to construct a formula θ_x defining x. For the rest of Section 3.3 we fix the delayed type x and assume that every delayed type $y \triangleleft x$ is definable by a formula θ_y.

The first case is when x has no neutral letters. Let $x-$ denote the set $\{y : y \triangleleft x\}$. By Lemma 13, in a tree of delayed type x both sons have delayed types in $x-$, since there are no neutral letters for x. In this case we can set

$$\theta_x = \mathit{fork}_x^{x-} \ . \tag{4}$$

The correctness of this definition follows immediately from Lemma 14.

The definition of θ_x is more involved when the set of neutral letters for x is not empty. The rest of Section 3.3 is devoted to this case.

Consider first the following formula:

$$\theta_{\not x} = \big(\mathsf{EF} \bigvee \{b \wedge \theta_y : y \triangleleft x \wedge (y,b) \ntrianglelefteq x\}\big) \vee \bigvee \{\mathit{fork}_y^{x-} : y \ntrianglelefteq x\}$$

The intention of this formula is to spell out evident cases when the delayed type of a node cannot be x. The first disjunct says that there is a descendant with a delayed type and a label that prohibit its ancestors to have type x. The second disjunct says that the type of the node is not x but the types of all descendants are $\trianglelefteq x$.

This formula works correctly only when some assumptions about the tree are made. These assumptions use the following definition: given a delayed type x, we define OK_x to be the set of tree t such that:

$$\mathit{dtype}(t) \lhd x \qquad \text{or} \qquad \mathit{dtype}(t) = x \text{ and } t(\epsilon) \in N_x \ .$$

Note that by Lemma 13 the set OK_x is closed under taking subtrees.

Lemma 15 Let t be a tree with the left and right subtrees in OK_x. This tree satisfies $\theta_{\not x}$ if and only if $\mathit{dtype}(t) \ntrianglelefteq x$.

Proof
The left to right implication was already discussed and follows from the assumptions on the θ_y formulas used in $\theta_{\not x}$ and from Lemma 14.

For the right to left implication, let $\mathit{dtype}(t) = \mathit{dtype}(y, b, z, c)$ with y, b, z, c describing delayed types and labels of the left and right subtrees of t. We consider three cases:

- $y = z = x$. This is impossible because both the left and right subtrees of t belong to OK_x, so the labels a, b must belong to N_x, and thus $\mathit{dtype}(t) = x$.

- $y = x$ and $z \lhd x$. Since the left subtree of t belongs to OK_x, the label b belongs to N_x. If the inequality $(z, c) \trianglelefteq x$ were true (which is not necessarily implied by our assumption that $z \lhd x$), then by property P3 we would have

  $$\mathit{dtype}(t) = \mathit{dtype}(y, b, z, c) = \mathit{dtype}(x, b, z, c) = \mathit{dtype}(x, b, x, b) = x \ ,$$

  a contradiction with $\mathit{dtype}(t) \ntrianglelefteq x$. Therefore we have $(z, c) \ntrianglelefteq x$ and hence the first disjunct of $\theta_{\not x}$ holds. The case where $z = x$ and $y \lhd x$ is symmetric.

- $y, z \lhd x$. In this case the second disjunct in the definition of $\theta_{\not x}$ must hold by Lemma 14.

$\square$

Let $\theta_{\lhd x}$ stand for $\bigvee_{y \lhd x} \theta_y$ and consider the formula

$$\varphi_x = \theta_{\lhd x} \vee (\neg\theta_{\not\trianglelefteq x} \wedge \bigvee\{a : a \in N_x\}) \ .$$

This formula will be used to define the set OK_x. We use AG^* as the non-strict version of AG, i.e. $\mathsf{AG}^*\varphi$ is an abbreviation for the formula $\varphi \wedge \mathsf{AG}\varphi$.

Lemma 16 A tree satisfies $\mathsf{AG}^*\varphi_x$ if and only if it belongs to OK_x.

Proof
By induction on the depth of the tree.

$\Rightarrow$ If t satisfies φ_x because it satisfies $\theta_{\lhd x}$, then it obviously belongs to OK_x. Otherwise we have

$$t(\epsilon) \in N_x \qquad \text{and} \qquad t \not\models \theta_{\not\trianglelefteq x} \ .$$

By induction assumption all proper subtrees of t belong to OK_x. But then, by Lemma 15, $\mathit{dtype}(t) \trianglelefteq x$. This, together with $t \not\models \theta_{\lhd x}$ shows that the delayed type of t is x and therefore t belongs to OK_x.

$\Leftarrow$ Let t be a tree in OK_x. By induction assumption we have $\mathsf{AG}\varphi_x$. We need to prove that t satisfies φ_x. If $\mathit{type}(t) \lhd x$ holds, then t satisfies $\theta_{\lhd x}$ and we are done. Otherwise, as $\mathrm{OK}_x(v)$ holds, $\mathit{dtype}(t) = x$ and $t(\epsilon) \in N_x$. Hence, by Lemma 15, t satisfies the second disjunct in φ_x.

$\square$

Since the type of a tree can be computed from its delayed type and root label, the following lemma ends the proof that every EF-admissible language is definable in EF:

Lemma 17 Every delayed type is definable.

Proof
By induction on the depth of a delayed type x in the order $\trianglelefteq$. If x has no neutral letters then the defining formula θ_x is as in (4). Otherwise, we set the defining formula to be

$$\theta_x = \neg\theta_{\lhd x} \wedge \neg\theta_{\not\trianglelefteq x} \wedge \mathsf{AG}\varphi_x \ .$$

Let us show why θ_x has the required properties. By Lemma 16,

$$t \models \mathsf{AG}\varphi_x \qquad \text{iff} \qquad t|_0, t|_1 \in \mathrm{OK}_x \ . \tag{5}$$

If $t \vDash \theta_x$ then we get $dtype(t) = x$ using Lemma 15 and (5). For the other direction, if $dtype(t) = x$ then clearly $\neg\theta_{\lhd x}$ holds in t. By Lemma 13 the left and right subtrees of t are in OK_x, therefore t satisfies $\mathsf{AG}\varphi_x$ by (5). But then the formula $\neg\theta_{\not\leq x}$ holds by Lemma 15. $\square$

3.4 A *fork* Formula

Recall Lemma 14 which was used in Section 3.3, but not proved there:

Lemma 14 Let x be a delayed type and let $A \not\ni x$ be a downward closed set of definable delayed types. There is a EF formula $fork_x^A$ such that:

$$t \vDash fork_x^A \quad \text{iff} \quad dtype(t) = x \text{ and for all } w > \epsilon,\ dtype(t|_w) \in A.$$

The rest of Section 3.4 is devoted to a proof of this lemma. We fix a delayed type x and a downward closed set of delayed types A. We assume that $x \notin A$ and that all the delayed types in A are definable.

For a delayed type $y \in A$ and a letter $b \in \Sigma$, we say that the pair (y, b) is *sufficient* if $dtype(y, b, y, b)$ is our fixed delayed type x. Given a delayed type y, we define the following equivalence relation $\sim_y$ over Σ:

$$a \sim_y b \qquad \text{iff} \qquad a = b \quad \text{or} \quad a, b \in N_y\ .$$

Remember that we want an EF formula expressing the fact that a tree t has delayed type x, though its proper subtrees only have smaller delayed types. Let y,b and z,c be the delayed types and root labels of the left and right subtrees of t. If the language is EF-admissible then there are essentially two possible reasons for this:

(i) The pair (y, b) is sufficient and $(z, c) \unlhd y$ holds (or the other way round);

(ii) Neither (y, b) nor (z, c) is sufficient but $dtype(y, b, z, c) = x$.

If we had the next modality EX, expressing the above properties would be very simple. Unfortunately this is not the case and we will need to use some rather complicated coding, which involves the following formulas:

$$\begin{aligned}
\theta_{\lhd y} &= \bigvee_{z \lhd y} \theta_z && \text{for } y \in A \cup \{x\}; \\
\theta_{\unlhd y} &= \bigvee_{z \unlhd y} \theta_z && \text{for } y \in A; \\
\theta_y^b &= \mathsf{EF}(\theta_y \wedge b) \wedge \mathsf{AG}(\theta_y \Rightarrow (b \vee N_y)) && \text{for } y \in A, b \in \Sigma; \\
\theta^{\unlhd y} &= \bigvee\{\theta_z \wedge c : (z,c) \unlhd y\} && \text{for } y \in A; \\
\varphi_y^b &= \theta_y^b \wedge \mathsf{AG}\theta_{\unlhd y} \wedge \mathsf{AG}(\theta_{\lhd y} \Rightarrow \theta^{\unlhd y}) && \text{for } y \in A, b \in \Sigma.
\end{aligned}$$

Observe that these formulas are well defined because we have assumed that all delayed types in A are definable, hence the appropriate θ_y formulas exist.

Lemma 18 If (y, b) is sufficient, a tree satisfying φ_y^b has delayed type x.

Proof
Let t be a tree that satisfies φ_y^b. First we will show that either the left or right subtree must have delayed type y and a root label $\sim_y$-equivalent to b. Let Y be the proper subtrees of t with delayed type y; this set is not empty because t satisfies $\mathsf{EF}(\theta_y \wedge b)$. Since $\unlhd$ is a partial order and $\mathsf{AG}\theta_{\unlhd y}$ holds, every tree with a subtree in Y also belongs to Y. If $b \notin N_y$ then by $\mathsf{EF}(\theta_y \wedge b)$, there is a tree in Y with the root labelled b. This must be either the left or right subtree of t as Lemma 13 says that all trees in Y apart from $t|_0$ and $t|_1$ must have labels in N_y. If the letter b belongs to N_y then, by $\mathsf{AG}(\theta_y \Rightarrow (b \vee N_y))$ the root label of every tree in Y is in N_y and hence $\sim_y$-equivalent to b. But this holds for either the left or right subtree, since Y is nonempty.

Now we can prove that t has delayed type x. By the reasoning above, a tree $s \in \{t|_0, t|_1\}$ has delayed type y and a root label $\sim_y$-equivalent to b. If both the left and right subtrees have delayed type y and labels $\sim_y$-equivalent to b then, by property P2, t is of delayed type x. If the brother of s has delayed type y but a root label c that is not $\sim_y$-equivalent to b then c must belong to N_y, because $\mathsf{AG}(\theta_y \Rightarrow (b \vee N_y))$ holds. By definition of N_y we have $(y, c) \unlhd y$. By property P3 we get $dtype(y, b, y, c) = dtype(y, b, y, b) = x$. Otherwise, by $\mathsf{AG}\theta_{\unlhd y}$, the brother of s is of delayed type $z \lhd y$ and has a root label c such that $(z, c) \unlhd y$ (because $\mathsf{AG}(\theta_{\lhd y} \Rightarrow \theta^{\unlhd y})$ holds). By property P3 $dtype(y, b, z, c) = dtype(y, b, y, b) = x$. $\square$

Given two delayed types $y, z \in A$ and letters $b, c \in \Sigma$, we define

$$\varphi_{(y,b,z,c)} = \begin{cases} \varphi_y^b & \text{if } (z,c) \trianglelefteq y; \\ \varphi_z^c & \text{if } (y,b) \trianglelefteq z \text{ and not the above;} \\ \theta_y^b \wedge \theta_z^c \wedge \mathsf{AG}(\theta_{\trianglelefteq y} \vee \theta_{\trianglelefteq z}) & \text{otherwise.} \end{cases}$$

Note that this formula corresponds to the two cases (i) and (ii) described before Lemma 18.

Lemma 19 If $dtype(y, b, z, c) = x$ and a tree t satisfies $\varphi_{(y,b,z,c)}$, then its delayed type is x.

Proof
If $(z, c) \trianglelefteq y$ then by property P3, the pair (y, b) is sufficient and the lemma follows from Lemma 18. Similarly if $(y, b) \trianglelefteq z$. It remains to consider the case when

$$(z, c) \ntrianglelefteq y \qquad \text{and} \qquad (y, b) \ntrianglelefteq z\ . \tag{6}$$

Let s_1 be a proper subtree satisfying $\theta_y \wedge b$ and s_2 be a proper subtree satisfying $\theta_z \wedge c$. These exist since t satisfies both θ_y^b and θ_z^c. Let t_1 be the son of t containing s_1, similarly we define t_2 for s_2. By (6), the tree t_1 does not satisfy $\theta_{\trianglelefteq z}$, while t_2 does not satisfy $\theta_{\trianglelefteq y}$. Hence it must be the case that

$$t_1 \models \theta_{\trianglelefteq y} \qquad \text{and} \qquad t_2 \models \theta_{\trianglelefteq z}\ .$$

In particular t_1 and t_2 are different subtrees. By a reasoning similar to the one in Lemma 18, one shows that the delayed type and root label of t_1 are y, b and the delayed type and root label of t_2 are z, c. By property P2, the delayed type of the tree t is x. $\square$

Lemma 20 Let y,b and z,c be the delayed types and root labels of the left and right subtrees of t. If $dtype(y, b, z, c) = x$ and $y, z \in A$ then $t \models \theta_{(y,b,z,c)}$.

Proof
If $(z, c) \trianglelefteq y$ then $z \trianglelefteq y$ and an easy analysis shows that t satisfies φ_y^b and hence also $\theta_{(y,b,z,c)}$. A similar reasoning shows that if $(y, b) \trianglelefteq z$ then t satisfies $\theta_{(y,b,z,c)}$. The last case is when $(z, c) \ntrianglelefteq y$ and $(y, b) \ntrianglelefteq z$. But then t satisfies the formula $\theta_y^b \wedge \theta_z^c \wedge \mathsf{AG}(\theta_{\trianglelefteq y} \vee \theta_{\trianglelefteq z})$. $\square$

But Lemmas 19 and 20 are exactly what we need to show that the *fork* formula defined below satisfies the properties postulated in Lemma 14:

$$fork_x^A = (\mathsf{AG} \bigvee_{y \in A} \theta_y) \wedge \bigvee \{\varphi_{(y,b,z,c)} : x = dtype(y, b, z, c)\}$$

4 Languages Definable in EX+EF

The last logic we consider in this paper is EX+EF. As in the previous sections, we will present a characterization of languages definable in EX+EF. For the rest of the section we fix an alphabet Σ along with a Σ-language L and will henceforth omit the L qualifier from notation.

Recall the type reachability quasiorder $\preccurlyeq$ along with its accompanying equivalence relation $\approx$, which were defined on p. 7. The $\approx$-equivalence class of a type α is called here its *component*. We extend the relation $\preccurlyeq$ to components by setting:

$$\begin{array}{lll} \Delta \preccurlyeq \Gamma & \text{if} & \alpha \preccurlyeq \beta \text{ for some } \alpha \in \Delta \text{ and } \beta \in \Gamma; \\ \alpha \preccurlyeq \Gamma & \text{if} & \alpha \preccurlyeq \beta \text{ for some } \beta \in \Gamma. \end{array}$$

We use the standard notational shortcuts, writing $\Delta \prec \Gamma$ when $\Delta \preccurlyeq \Gamma$ but not $\Gamma = \Delta$; similarly for $\alpha \prec \Gamma$.

Let Γ be some component and let $k \in \mathcal{N}$. The (Γ, k)-*view* of a tree t is the tree $\mathit{view}(\Gamma, k, t)$ whose domain is the set of nodes in t at depth at most k and where a node v is labeled by:

- $t(v)$ if v is at depth smaller than k;
- $\mathit{type}(t|_v)$ if v is at depth k and $\mathit{type}(t|_v) \prec \Gamma$;
- ? otherwise.

Let $\mathit{views}(\Gamma, k)$ denote the set of possible (Γ, k)-views. The intuition behind the (Γ, k)-view of t is that it gives exact information about the tree t for types which are $\prec$ smaller than Γ, while for other types it just says "I don't know". The following definition describes languages where this information is sufficient to pinpoint the type within the strongly connected component Γ.

Definition 21 Let Γ be a component and let $k \in \mathcal{N}$. The language L is (Γ, k)-solvable if every two trees s and t with types in Γ and the same (Γ, k) view have the same type. The language is k*-solvable* if it is (Γ, k)-solvable for every component Γ and it is *component solvable* if it is k-solvable for some k.

It turns out that component solvability is exactly the property which characterizes the languages definable in EX+EF:

Theorem 22
A regular language is definable in EX+EF if and only if it is component solvable.

The proof of both implications in this theorem will be presented in the two subsections that follow.

4.1 A Component Solvable Language Is Definable in EX+EF

In this section we show that one can write EX+EF formulas which compute views. Then, using these formulas and the assumption that L is component solvable, the type of a tree can be found.

Fix some k such that L is k-solvable. Let α be a type in a component Γ. We write $\mathit{views}(\alpha)$ to denote set of possible (Γ, k)-views that can be assumed in a tree of type α. By assumption on L being k-solvable, we have:

Fact 23 Let α be a type in component Γ and let t be a tree such that $\mathit{type}(t) \preccurlyeq \alpha$. The type of t is α if and only if its (Γ, k)-view belongs to the set $\mathit{views}(\alpha)$.

The following lemma states that views can be computed in EX+EF.

Lemma 24 Suppose that for every type $\beta \prec \Gamma$, there is a EX+EF formula θ_β defining it. Then for every $i \in \mathcal{N}$ and every $s \in \mathit{views}(\Gamma, i)$ there is a formula ψ_s satisfied in exactly the trees whose (Γ, i)-view is s.

Proof
By induction on i. $\square$

We define below a set of views which certainly cannot appear in a tree with a type in a strongly connected component Γ:

$$\begin{aligned} \mathrm{Bad}(\Gamma) \quad = \quad & \{a[s,t] : s \in \mathit{views}(\alpha), t \in \mathit{views}(\beta), \text{where } \alpha, \beta \preccurlyeq \Gamma, a[\alpha, \beta] \not\preccurlyeq \Gamma\} \cup \\ & \cup \{t : \mathit{type}(t) \not\preccurlyeq \Gamma \text{ and } \mathrm{dom}(t) = \{\epsilon\}\} \end{aligned}$$

Observe that $\mathrm{Bad}(\Gamma)$ is a set of $(\Gamma, k+1)$-views. The following lemma shows that the above cases are essentially the only ones.

Lemma 25 For a tree t and a component Γ, the following equivalence holds:

$$\mathit{type}(t) \not\preccurlyeq \Gamma \quad \text{iff} \quad \mathit{view}(\Gamma, k+1, t|_v) \in \mathrm{Bad}(\Gamma) \text{ for some } v \in \mathrm{dom}(t).$$

Proof
Both implications follow easily from Fact 23 if one considers the maximal possible node v satisfying the right hand side. $\square$

The following lemma completes the proof that L is definable in EX+EF.

Lemma 26 Every type of L is definable in EX+EF.

Proof
The proof is by induction on depth of the type in the quasiorder $\preccurlyeq$. Consider a type α and its component Γ. By induction assumption, for all types $\beta \prec \Gamma$, there is a formula θ_β which is satisfied in exactly the trees of type β. Using the θ_β formulas and Lemma 24 we construct the following EX+EF formula (recall that AG^* is the non-strict version of AG defined on page 13):

$$\theta_\Gamma = \mathsf{AG}^* \bigwedge_{t \in \mathrm{Bad}(\Gamma)} \neg\psi_t.$$

By Lemma 25, a tree t satisfies θ_Γ if and only if $\mathit{type}(t) \preccurlyeq \Gamma$. Once we know that the component of the tree is Γ, we can use Fact 23 to pinpoint the exact type:

$$\theta_\alpha = \theta_\Gamma \wedge \bigvee_{t \in \mathit{views}(\alpha)} \psi_t.$$

$\square$

4.2 A Language Definable in EX+EF is Component Solvable

In this section, we are going to show that a language which is not component solvable is not definable in EX+EF. For this, we introduce an appropriate Ehrenfeucht-Fraïsé game, called the *EX+EF game*, which characterizes trees indistinguishable by EX+EF formulas.

The game is played over two trees and by two players, Spoiler and Duplicator. The intuition is that in the *k-round EX+EF game*, the player Spoiler tries to differentiate the two trees using k moves.

The precise definition is as follows. At the beginning of the k-round game, with $k \geq 0$, the players are faced with two trees t_0 and t_1. If these have different root labels, Spoiler wins. If they have the same root labels and $k = 0$, Duplicator wins; otherwise the game continues. Spoiler first picks one of the trees t_i, with $i \in \{0, 1\}$. Then he chooses whether to make an EF or EX move. If he chooses to make EF move, he needs to choose some non-root node $v \in \mathrm{dom}(t_i)$ and Duplicator must respond with a non-root node $w \in \mathrm{dom}(t_{1-i})$ of the other tree. If Spoiler chooses to make an EX move, he picks a son $v \in \{0, 1\}$ of the root in t_i and Duplicator needs to pick the same son $w = v$ in the other tree. If a player cannot find an appropriate node in the relevant tree, this player immediately looses. Otherwise the trees $t_i|_v$ and $t_{1-i}|_w$ become the new position and the $(k-1)$-round game is played.

The *modality nesting depth* of a formula is defined by induction in the natural fashion. Formulas that correspond to letters have depth zero, the depth of a boolean combination is the maximal depth of the formulas involved, while applying EX or EF to a formula increases the depth by one.

Lemma 27 Duplicator wins the k-round EX+EF game over t_0 and t_1 iff t_0 and t_1 satisfy the same EX+EF formulas of modality nesting depth k.

Proof
A standard proof by induction on k. The case of $k = 0$ is obvious. Let us assume that we have proved the statement for some k and consider $k + 1$.

Consider first the left to right implication. We show that if a formula φ distinguishes the trees t_0 and t_1, then a winning strategy for Spoiler can be found. If φ distinguishes the trees t_0 and t_1, then one of its subformulas of the form $\mathsf{EX}\psi$ or $\mathsf{EF}\psi$ distinguishes them too. Let us consider the case of $\mathsf{EF}\psi$ and assume without loss of generality that $\mathsf{EF}\psi$ holds only in t_0. This means that there is a nonroot node v_0 in the tree t_0 such that

$$t_0|_{v_0} \models \psi \qquad \text{and} \qquad t_1|_{v_1} \not\models \psi \text{ for all nonroot nodes } v_1 \text{ of } t_1$$

The winning strategy for Spoiler is, of course, to pick an EF move, the tree t_0 and the vertex v_0. Since ψ is of modality nesting depth k, no matter what vertex v_1 Duplicator picks, Spoiler has – by induction assumption – a winning strategy in the k-round game over the trees $t_0|_{v_0}$ and $t_1|_{v_1}$. A similar argument is used when the distinguishing formula is of the form $\mathsf{EX}\psi$.

For the right to left implication, we show how to write a distinguishing formula ψ of nesting depth $k+1$ based on the assumption that Spoiler wins the $k+1$-round game. Consider a winning strategy of Spoiler in this game. We assume without loss of generality that Spoiler chooses the tree t_0 to make his move. Two cases need be considered. The first is when Spoiler chooses an EF move and a subtree $t_0|_{v_0}$. Since his strategy is winning, for every possible choice of a node v_1 in the tree t_1, the k-round game over the trees $t_0|_{v_0}$ and $t_1|_{v_1}$ can be won by Spoiler. By induction assumption this means that for every node v_1 in the tree t_1, there is a formula ψ_{v_1} of nesting depth k such that

$$t_0|_{v_0} \models \psi_{v_1} \qquad \text{and} \qquad t_1|_{v_1} \not\models \psi_{v_1} \ .$$

Note that in order to have ψ_{v_1} satisfied in $t_0|_{v_0}$ and not in $t_1|_{v_1}$, we may have negated the formula from the induction assumption. Let ψ be a conjunction of all the ψ_{v_1} formulas for all choices of v_1. The appropriate formula that distinguishes the trees t_0 and t_1 is then $\mathsf{EF}\psi$. A similar reasoning is used for EX. $\square$

A *multicontext* is like a context but it may have more than one hole. Given a multicontext C and a function ν which to every hole in C assigns a tree, we define the tree $C[\nu]$ in the natural way. Similarly we proceed when ν assigns types instead of trees: in this case $C[\nu]$ is a type. The *hole depth* of a multicontext C is the minimal depth of a hole in C.

For two types α, β in a component Γ we define an *(α, β)-context* to be a multicontext C with holes V such that there are two valuations $\nu_\alpha, \nu_\beta : V \to \Gamma$ giving the types $C[\nu_\alpha] = \alpha$ and $C[\nu_\beta] = \beta$. A multicontext C is *k-bad for component* Γ if it has hole depth at least k and is an (α, β)-context for two different types $\alpha, \beta \in \Gamma$. The following Lemma shows that (α, β)-contexts are just a reformulation of component solvability:

Lemma 28 *L is not component solvable if and only if for some component Γ and every $k \in \mathcal{N}$, it contains multicontexts which are k-bad for Γ.*

Proof
A k-bad context exists for Γ if and only if L is not (Γ, k)-solvable. $\square$

The following lemma concludes the proof that no EX+EF formula can recognize a language which is not component solvable:

Lemma 29 *If L is not component solvable then for every k there are trees $s \in L$ and $t \notin L$ such that Duplicator wins the k-round EX+EF game over s and t.*

Proof
Take some $k \in \mathcal{N}$. If L is not component solvable then, by Lemma 28, there is a multicontext C which is k-bad for some component Γ. Let $V = \{v_1, \ldots, v_n\}$ be the holes of C, let $\nu_\alpha, \nu_\beta : V \to \Gamma$ be the appropriate valuations and $\alpha = C[\nu_\alpha], \beta = C[\nu_\beta]$ the resulting types. We will use this multicontext to find trees $s \in L$ and $t \notin L$ such that Duplicator wins the k-round EX+EF game over s and t.

Since all the types used in the valuations ν_α and ν_β are from same component, there are contexts $C_1^\alpha[], \ldots, C_n^\alpha[]$ and $C_1^\beta[], \ldots, C_n^\beta[]$ such that

$$C_i^\alpha[\alpha] = \nu_\beta(v_i) \qquad C_i^\beta[\beta] = \nu_\alpha(v_i) \qquad \text{for all } i \in \{1, \ldots, n\}.$$

This means there are two contexts D^α and D^β with n holes each, such that: 1) D^α and D^β agree over nodes of depth less than k; 2) when all holes of D^α are plugged with β, we get the type α; and 3) when all holes of D^β are plugged with α, we get the type β. These are obtained by plugging the appropriate "translators" $C_i^\alpha[]$ and $C_i^\beta[]$ into the holes of the multicontext C. Let t_0 be some tree of type α. The trees t_j for $j > 0$ are defined by induction as follows:

$$t_{2i+1} = D^\beta[\overbrace{t_{2i}, \ldots, t_{2i}}^{n \text{ times}}] \qquad t_{2i+2} = D^\alpha[\overbrace{t_{2i+1}, \ldots, t_{2i+1}}^{n \text{ times}}].$$

By an obvious induction, all the trees t_{2i} have type α and all the trees t_{2i+1} have type β. As $\beta \neq \alpha$, there exists a context $D[]$ such that $D[\alpha] \in L$ and $D[\beta] \notin L$ (or the other way round).

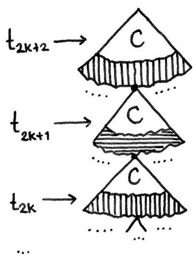

Figure 2: The tree t_{2i+2}.

To finish the proof of the lemma, we will show that Duplicator wins the k-round EX+EF game over the trees

$$s = D[t_{2k+2}] \qquad \text{and} \qquad t = D[t_{2k+1}] \ .$$

The winning strategy for Duplicator is obtained by following an invariant. This invariant is a disjunction of three properties, one of which always holds when the i-round game is about to be played:

1. The two trees are identical;
2. The two trees are $s|_v$ and $t|_v$ for some $|v| \leq k - i$;
3. The two trees are $t_m|_v$ and $t_{m-2}|_v$ for

$$m \geq k + i + 1 \quad \text{and} \quad \begin{cases} v \in \mathrm{dom}(D^\alpha) & \text{if } m \text{ is even;} \\ v \in \mathrm{dom}(D^\beta) & \text{if } m \text{ is odd.} \end{cases}$$

The invariant holds at the beginning of the first round, due to 2, and one can verify that Duplicator can play in such a way that it is satisfied in all rounds. Item 2 of the invariant will be preserved in the initial fragment of the game when only EX moves are made, then item 3 will hold until either the game ends or item 1 begins to hold. □

5 Decidability

In this section we round up the results by showing that our characterizations are decidable.

Theorem 30
It is decidable in time polynomial in the number of types if a language is definable in any one of the logics EX, EF or EX+EF.

Proof
Using a simple dynamic algorithm, one can compute in polynomial time all tuples $(\alpha, \beta, \alpha', \beta')$ such that for some context $C[]$, $C[\alpha] = \alpha'$ and $C[\beta] = \beta'$. Using this, we can find in polynomial time:

- Whether L contains an $\{\alpha, \beta\}$-loop;
- The $\preccurlyeq_L$ and $\approx_L$ relations on types.

Since the delayed type of a tree depends only on the types of its immediate subtrees, the number of delayed types is polynomial in the number of types. The relation $\trianglelefteq_L$ on delayed types can then be computed in polynomial time from the relation $\preccurlyeq_L$. Having the relations $\preccurlyeq_L$ and $\trianglelefteq_L$, one can check in polynomial time if L is EF-admissible.

This, along with the characterizations from Theorems 3 and 5, proves decidability for the logics EX and EF. The remaining logic is EX+EF.

By Theorem 22, it is enough to show that component solvability is decidable. In order to do this, we present an algorithm that detects if a given component Γ admits bad multicontexts of arbitrary size, cf. Lemma 28. Fix a component Γ. We define by induction a sequence B^i of subsets of $\Gamma \times \Gamma$.

- B^0 consisits of all pairs (α, β) such that $\alpha, \beta \in \Gamma$ and $\alpha \neq \beta$.
- A pair (α, β) belongs to B^{i+1} if it belonged to B^i and either
  - there are a pair $(\alpha', \beta') \in B^i$, a type $\gamma \preccurlyeq \Gamma$ and a letter $a \in \Sigma$ such that

$$\mathit{type}(a[\alpha', \gamma]) = \alpha \qquad \text{and} \qquad \mathit{type}(a[\beta', \gamma]) = \beta \text{ ; or}$$

– there are pairs $(\alpha', \beta'), (\alpha'', \beta'') \in B^i$ and a letter $a \in \Sigma$ such that

$$\mathit{type}(a[\alpha', \alpha'']) = \alpha \qquad \text{and} \qquad \mathit{type}(a[\beta', \beta'']) = \beta \ .$$

The sequence B^i is decreasing so it reaches a fix-point B^∞ in no more than $|\Gamma|^2$ steps. The following lemma yields the algorithm for EX+EF:

Lemma 31 Γ admits bad multicontexts of arbitrary size iff $B^\infty \neq \emptyset$.

For the left-to-right implication suppose that B^∞ is not empty. By induction on k we show that for every k and $(\alpha, \beta) \in B^\infty$, we can construct (α, β)-context of hole depth k. Take $(\alpha, \beta) \in B^\infty$. We have one of the two cases from the definition above. The first is when there are a pair $(\alpha', \beta') \in B^\infty$, a type $\gamma \preccurlyeq \Gamma$ and a letter $a \in \Sigma$ such that $\mathit{type}(a[\alpha', \gamma']) = \alpha$ and $\mathit{type}(a[\beta', \gamma']) = \beta$. By induction assumption we have an (α', β')-context C' of hole depth $k-1$. Using this multicontext, we construct the multicontext $a[C', s]$, where s is a tree of type γ. It is a required (α, β)-context of hole depth k. The other case is similar.

For the right-to-left implication we show that if $(\alpha, \beta) \in B^i - B^{i+1}$ then all (α, β)-contexts have hole depth bounded by i. This is also done by induction on i.

□

Corollary 32 If the input is a CTL formula or a nondeterministic tree automaton, all of the problems in Theorem 30 are EXPTIME-complete.

Proof
Since, in both cases, the types can be computed in time at most exponential in the input size, the EXPTIME membership follows immediately from Theorem 30. For the lower bound, we will use an argument analogous to the one in [17], reducing the EXPTIME-hard universality problems for both CTL [3] and nondeterministic automata [13] to any of these problems.

We will only show here the EXPTIME-hardness of the problem:

Is a given CTL formula equivalent to one in EF? (*)

Let ψ be a CTL formula over some alphabet Σ. By [3], the question whether ψ is satisfied in all Σ-trees is EXPTIME-hard. We show the EXPTIME-hardness of the problem (*) by presenting a formula φ which is definable in the logic EF if and only if the formula ψ is true in all Σ-trees. This formula is obtained by using ψ and some fixed formula ϕ – say, $\mathsf{E}(a\mathsf{U}b)$ – not definable in EF:

$$\varphi = \mathsf{EX}_0\psi \vee \mathsf{EX}_1\phi$$

Clearly if ψ is true in all Σ-trees, then, by its first disjunct, φ is true in all Σ-trees with more than one node. This language is defined by the EF formula $\mathsf{EX}\top$.

Finally, we need to prove that if ψ is not true in all Σ-trees, then φ is not definable in EF. Let us assume for the sake of contradiction that φ is equivalent to some EF formula θ. Let Ψ be all the subformulas of θ. By assumption that the formula ϕ is not definable in EF, there exist two trees t_1 and t_2 that satisfy the same formulas in Ψ, but one satisfies ϕ and the other does not (otherwise an appropriate boolean combination of formulas in Ψ would be equivalent to ϕ). Therefore exchanging t_1 with t_2 in any subtree does not affect the satisfaction of θ. Let s be a tree that does not satisfy ψ, obtained by the assumption on ψ not being satisfied in all Σ-trees. Obviously for any letter $a \in \Sigma$ we have

$$a[s, t_1] \models \varphi \qquad a[s, t_2] \not\models \varphi \ ,$$

but either both these trees satisfy θ or both do not. □

6 Why Forbidden Patterns from the Word Case Do not Work

In the survey [18], one can find decidable characterizations for several fragments of LTL. These fragments can be seen as the word equivalents of the logics EX, EF and EX+EF considered here. One naturally asks the question: how are the word and tree cases related? In the case of the logic EX, the loop characterization from Theorem 3 is an exact analogue of the characterization corresponding to the fragment of LTL that only uses the X modality.

For the two remaining logics, however, the word and tree cases diverge. This section is devoted to showing why.

Case of EF. First we need to introduce the appropriate definitions for words. The delayed type of a word w is the function which assigns to a letter a the type of the word $a \cdot w$. Two word types α, β are in the same component if there are word types α', β' such that $\alpha' \cdot \alpha = \beta$ and $\beta' \cdot \beta = \alpha$. In [18] it is shown that a word language is definable using only the modality

F if and only if the delayed type of a word is determined by the component of its type.

Hence a natural question: is a tree language definable in EF if and only if the delayed type of a tree is determined by the components of its two sons? This can be understood in two ways: the ordered pair of components, or the set of components. The first idea can be immediately disproved, for instance using the language "there is an a in the left subtree". The idea that uses sets requires a more elaborate example, which is presented here.

	a	b
$x_\top$	$\top$	$\top$
x_a	α	γ
x_b	γ	β
x_γ	γ	γ

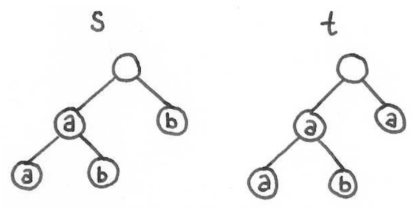

Figure 3: The delayed types of L along with the trees s and t

Consider the $\{a, b\}$-language L defined by the formula (which uses the non-strict versions of AG and EF):

$$\psi = \mathsf{EF}^*[\mathsf{EX}(\mathsf{AG}^* a) \wedge \mathsf{EX}((\mathsf{EF}^* a) \wedge (\mathsf{EF}^* b))]$$

A tree t satisfies this formula if it contains two nodes v and w which are siblings and $t|_v$ contains only a's, while $t|_w$ contains both a's and b's. This language has four types α, β, γ and $\top$, which are defined by the formulas:

$$\alpha = \mathsf{AG}^* a; \quad \beta = \mathsf{AG}^* b; \quad \gamma = (\mathsf{EF}^* a) \wedge (\mathsf{EF}^* b) \wedge \neg\psi; \quad + = \psi.$$

Each of these types is its own component. There are also four delayed types $\{x_+, x_a, x_b, x_\gamma\}$, which are defined in Figure 3.

Consider now the two trees s and t drawn in Figure 3 (we do not specify the root letters since we are interested only in delayed types). These trees show that the language L is not typeset dependent and therefore not in EF, since:

$$\mathrm{TS}(s) = \mathrm{TS}(t) = \{\alpha, \beta, \gamma\} \quad \text{but} \quad x_\gamma = \mathit{dtype}(s) \neq \mathit{dtype}(t) = x_+.$$

However, if the components of both sons are known, then the types of both sons are known and hence so is the delayed type of the tree.

Case of EX+EF. In [18] it is shown that a word language is definable using the modalities F and X if and only if one cannot find two distinct types α, β in the same component and a nonempty word w such that:

$$\alpha = w \cdot \alpha \quad \text{and} \quad \beta = w \cdot \beta. \tag{7}$$

We will show that a straightforward generalization of this condition obtained by considering contexts instead of words does not work in the tree case. Consider the language K over the alphabet $\{a, b, c\}$ defined by the formula

$$\psi = \mathsf{AG}^* \bigvee_{\sigma \neq \tau \in \{a,b,c\}} \psi_{\sigma,\tau} \qquad \text{where} \quad \psi_{\sigma,\tau} = [\mathsf{EX}\top \Rightarrow (\sigma \wedge \mathsf{A}(\sigma \mathsf{U} \tau))] \ .$$

This language consists of those trees where for every node v, all the minimal nodes in the set $\{w : w > v \text{ and } t(w) \neq t(v)\}$ have the same label. The ten types of the language are:

$$0 = \neg\psi; \qquad \alpha_\sigma = \sigma \wedge \mathsf{AX}\bot; \qquad \beta_{\sigma,\tau} = \psi \wedge \psi_{\sigma,\tau} \qquad \text{for all } \sigma \neq \tau \in \{a, b, c\}.$$

There are five components in this language: a component Γ containing the six $\beta_{\sigma,\tau}$ types, while each of the remaining types is its own component.

The language K is not k-solvable for any $k \in \mathcal{N}$, and hence is not definable in EX+EF. We will show, however, that if in the condition (7) one considers nontrivial contexts instead of nonempty words w, the resulting condition on tree languages is satisfied by K. This goes to show that in a bad multicontext one sometimes needs the use of more than one hole.

Let $C[]$ be a nontrivial context, i.e. one with the hole not in the root. We will show that one cannot find two distinct types in the component Γ such that

$$C[\gamma_1] = \gamma_1 \qquad \text{and} \qquad C[\gamma_2] = \gamma_2. \tag{8}$$

Let σ be the letter in the parent v of the hole, and let w be the brother of the hole. Let $V = \{u : u \geq w \text{ and } C(u) \neq \sigma\}$. If $V = \emptyset$, or nodes in V have two different labels, then $C[\gamma] = 0$ for all types γ. Otherwise, let τ be the unique label of all nodes in V. This means that for any tree t, the type in v of $C[t]$ is either 0 or $\beta_{\sigma,\tau}$, which proves that (8) cannot be satisfied.

7 Open Problems

This paper solves the question of definability for the logics EX, EF and EX+EF. One possible continuation are logics where instead of EF we use the non-strict modality EF^*. The resulting logics are weaker than their strict counterparts (for instance the language $\mathsf{EF}a$ is not definable using only EF^*) and therefore decidability of the their definability problems can be investigated.

Another question is what happens if we enrich these logics with past quantification (there exists a point in the past)? This question is particularly relevant since the resulting logics are related to first-order logic with two variables.

Finally, there is CTL (and of course CTL*). Providing a decidable characterization of CTL would be a valuable achievement, since this is a widely used logic. Note that on words CTL collapses to LTL and hence first-order logic, so such a characterization would subsume first-order definability for words.

References

[1] E. A. Emerson and J. Y. Halpern. 'Sometimes' and 'not never' revisited: on branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[2] J. Esparza. Decidability of model-checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.

[3] M. Fischer and R. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.

[4] M. Franceschet, L. Afanasiev, M. de Rijke, and M. Marx. CTL model checking for processing simple XPath queries. In *Temporal Presentation and Reasoning*.

[5] C. Koch G. Gottlob. Monadic queries over tree-structured data. In *Logic in Computer Science*, pages 189–202, 2002.

[6] T. Hafer and W. Thomas. Computation tree logic CTL and path quantifiers in the monadic theory of the binary tree. In *International Colloquium on Automata, Languages and Programming*, volume 267 of *Lecture Notes in Computer Science*, pages 260–279, 1987.

[7] U. Heuter. First-order properties of trees, star-free expressions, and aperiodicity. In *Symposium on Theoretical Aspects of Computer Science*, volume 294 of *Lecture Notes in Computer Science*, pages 136–148, 1988.

[8] F. Neven. Automata, logic, and XML. In Julian C. Bradfield, editor, *Computer Science Logic*, volume 2471 of *Lecture Notes in Computer Science*, pages 2–26, 2002.

[9] D. Niwiński. Fixed points vs. infinte generation. In *Logic in Computer Science*, pages 402–409, 1988.

[10] M. Otto. Eliminating recursion in the μ-calculus. In *Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 531–540, 1999.

[11] A. Potthoff. First-order logic on finite trees. In *Theory and Practice of Software Development*, volume 915 of *Lecture Notes in Computer Science*, pages 125–139, 1995.

[12] M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

[13] H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal of Computing*, 19:424–437, 1990.

[14] H. Straubing. *Finite Automata, Formal Languages, and Circuit Complexity.* Birkhäuser, Boston, 1994.

[15] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[16] I. Walukiewicz. Model checking CTL properties of pushdown systems. In *Foundations of Software Technology and Theoretical Computer Science*, volume 1974 of *Lecture Notes in Computer Science*, pages 127–138, 2000.

[17] I. Walukiewicz. Deciding low levels of tree-automata hierarchy. In *Workshop on Logic, Language, Information and Computation*, volume 67 of *Electronic Notes in Theoretical Computer Science*, 2002.

[18] T. Wilke. Classifying discrete temporal properties. In *Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 32–46, 1999.