A FIRST-ORDER COMPLETE TEMPORAL LOGIC FOR STRUCTURED CONTEXT-FREE LANGUAGES

MICHELE CHIARI [a], DINO MANDRIOLI [a], AND MATTEO PRADELLA [a,b]

[a] DEIB, Politecnico di Milano, Milano, Italy
e-mail address: michele.chiari@polimi.it, dino.mandrioli@polimi.it, matteo.pradella@polimi.it

[b] IEIIT, Consiglio Nazionale delle Ricerche, Milano, Italy

Abstract. The problem of model checking procedural programs has fostered much research towards the definition of temporal logics for reasoning on context-free structures. The most notable of such results are temporal logics on Nested Words, such as CaRet and NWTL. Recently, the logic OPTL was introduced, based on the class of Operator Precedence Languages (OPLs), more powerful than Nested Words. We define the new OPL-based logic POTL and prove its FO-completeness. POTL improves on NWTL by enabling the formulation of requirements involving pre/post-conditions, stack inspection, and others in the presence of exception-like constructs. It improves on OPTL too, which instead we show not to be FO-complete; it also allows to express more easily stack inspection and function-local properties. In a companion paper we report a model checking procedure for POTL and experimental results based on a prototype tool developed therefor. For completeness a short summary of this complementary result is provided in this paper too.

1. Introduction

Linear Temporal Logic (LTL) is one of the most successful languages for the specification and verification of system requirements. Being defined on a linearly ordered algebraic structure, it is ideal to express safety and liveness properties on a linear flow of events. In particular, LTL is equivalent to First-Order Logic (FOL) on this structure [Kam68] and captures the first-order definable fragment of (ω-)regular languages. Its satisfiability and model checking are PSPACE-complete with respect to formula length but polynomial in model size.

The need for model checking software programs lead to several attempts at widening the expressive power of specifications beyond LTL. Indeed, procedural programs are often modeled with operational formalisms such as Boolean programs [BR00], Pushdown Systems, and Recursive State Machines [ABE$^+$05]. Reachability and LTL model checking have been extensively studied for such systems [BEM97, EHRS00, ABE$^+$05]. However, they present behaviors typical of pushdown automata. Thus, many properties thereof cannot be expressed by a formalism whose expressive power is limited to regular languages. Some early attempts at extending the expressive power of specifications beyond regular languages in model checking include equipping LTL with Presburger arithmetic [BH96], using directly a class of

Key words and phrases: Linear Temporal Logic, Operator-Precedence Languages, Model Checking, First-Order Completeness, Visibly Pushdown Languages, Input-Driven Languages.

Preprint submitted to
Logical Methods in Computer Science

pushdown automata for the specification [KPV02], and a variant of Propositional Dynamic Logic that can express a restricted class of Context-Free Languages (CFLs) [HKT02].

A natural way of generalizing LTL is defining temporal logics on more complex algebraic structures. The introduction of logics based on *Nested Words* [AM09], such as CaRet [AEM04], has been one of the first attempts in this direction. Nested Words consist of a discrete, linear order with the addition of a one-to-one nesting relation. They are equivalent to Visibly Pushdown Languages (VPLs) [AM04], a.k.a. Input-Driven Languages [Meh80], a strict subclass of deterministic CFLs. The parenthetical nature of VPLs results in the nesting relation on words, which naturally models the one-to-one correspondence between function calls and returns in procedural programs. CaRet is the first temporal logic that extends LTL through explicit modalities that refer to the underlying model's context-free structure. Thus, CaRet can easily express Hoare-style pre/post-conditions, stack inspection, and more.

The applications of Nested Words in model checking lead to the question of which theoretical properties of LTL could be extended to them. One of the most prominent is First-Order (FO) completeness. CaRet was devised with the specification of procedural programs in mind, but its stance with respect to FOL remains unknown, although it is conjectured not to be FO-complete [AAB$^+$08]. FO-completeness on Nested Words is thoroughly studied in [AAB$^+$08], and is achieved with Nested Words Temporal Logic (NWTL) and other logics. NWTL satisfiability and model checking are EXPTIME-complete and retain polinomiality on system size.

One natural further question is whether it is possible to define temporal logics that capture a fragment of CFLs wider than VPLs. Several practical applications motivate this question: the nesting relation of Nested Words is one-to-one [MP18], preventing its ability to model behaviors in which a single event is related to multiple ones. Such behaviors occur, e.g., in widespread programming constructs such as exceptions, interrupts and continuations.

OPTL [CMP20] is a recent temporal logic that can express specifications concerning, e.g., whether a function is terminated by an exception, or throws one, and pre/post-conditions. OPTL is based on Operator Precedence Languages (OPLs), which are a subclass of deterministic CFLs initially introduced with the purpose of efficient parsing [Flo63], a field in which they continue to offer useful applications [GJ08, BCM$^+$15]. They strictly contain VPLs [CM12] and retain all complexity, closure and decidability properties that make regular languages and VPLs well-suited for temporal logics: they are closed under Boolean operations, concatenation, Kleene *, and language emptiness and inclusion are decidable [CM12]. They have also been characterized through Monadic Second-Order Logic (MSO) [LMPP15]. This greater expressive power results in OPTL being based on a structure called *OP word*, made of a linear order plus a nesting relation that can be one-to-many and many-to-one, and considerably generalizes that of Nested Words. Indeed, OPTL is strictly more expressive than NWTL, but retains the same satisfiability and model checking complexity [CMP20]. However, the relationship between OPTL and FOL remained unknown.

In this paper, we close this gap thanks to the novel temporal logic POTL (Precedence-Oriented Temporal Logic). POTL too is based on OPLs and is devised to easily navigate a word's underlying syntax tree (ST). It is also based on OP words, enabling reasoning on constructs such as exceptions, but has a better ability to express properties on the context-free structure of words. It features new until and since modalities based on *summary* paths, which can navigate a word's ST up or down while skipping subtrees, and *hierarchical*

paths, which move between word positions where the nesting relation is many-to-one or one-to-many.

The gap in expressive power between OPTL and POTL is both practical and theoretical. In fact, in POTL it is easier to express stack inspection properties in the presence of uncaught exceptions, as well as function-frame local properties. We show that one of such properties is not expressible at all in OPTL, but it is in POTL, and hence POTL is strictly more expressive than OPTL.

Moreover, we prove that POTL is equivalent to FOL on both OP finite and ω-words. We obtain the result on finite words by translating an FO-complete logic for finite trees [Mar04] to POTL. We then extend the proof to ω-words by employing a composition argument on trees proved with Ehrenfeucht-Fraïssé games. This last proof is rather involved, as it needs to distinguish between two classes of trees that arise from OP ω-words.

As a corollary, we show that FOL has the three-variable property on OP words. I.e., every FO formula on OP words is equivalent to another FOL formula in which only three distinct variables appear. This property is related to FO-completeness in temporal logics [Gab81] and holds both on simple Dedekind-complete linear orders and Nested Words [AAB$^+$08].

Recently, we also showed [MPC20b, MPC20a] that FO-definable OPLs coincide with aperiodic or noncounting ones, according to a definition of aperiodic structured CFLs which naturally extends the classic one for regular languages [MP71][1]. Whereas various regular languages of practical usage are counting —e.g., many hardware devices are just counters modulo some integer— we are not aware of programming languages or other CFLs of practical interest, such as structured data description languages, that exhibit counting properties: e.g., no language imposes to write programs with an even number of nested loops. Thus, the breadth of coverage in terms of practical applications of model-checking algorithms for an FO-complete logic is even larger for the class of CFLs than that of classic model checkers for regular languages.

POTL's expressiveness gains do not come at the cost of higher model checking complexity, which remains EXPTIME-complete, as we show in [CMP21].

The paper is organized as follows: Section 2 provides some background on OPLs; Section 3 presents the syntax and semantics of POTL, with some qualitative demonstration of its expressive power and comparison with similar logics, and the proof of the expressive *in*completeness of OPTL; Section 4 proves the equivalence of POTL to FOL on finite words; Section 5 extends this result to ω-words; Section 6 summarizes the results of the companion paper [CMP21] where a model checker for OPLs based on POTL is provided, its complexity evaluated, and experimental results are also reported; Section 7 concludes and proposes future research lines. Appendices A and B contain proofs that did not fit into the main text.

We assume familiarity with classic topics of theoretical computer science, specifically context-free grammars and languages, pushdown automata, parsing, syntax tree (ST) (cf. [Har78]), ω-languages, i.e., languages with infinite words, and temporal logic —LTL in particular— [Eme90].

2. Operator Precedence Languages

Operator Precedence Languages (OPLs) were originally defined through their generating grammars [Flo63]: *operator precedence grammars* (OPGs) are a special class of context-free

[1]Noticeably, this equivalence does not hold for tree-languages and their FO-logic [Tho84, Heu91].

	call	**ret**	**han**	**exc**
call	$\lessdot$	$\doteq$	$\lessdot$	$\gtrdot$
ret	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
han	$\lessdot$	$\gtrdot$	$\lessdot$	$\doteq$
exc	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$

Figure 1: The OPM $M_{\mathbf{call}}$.

grammars (CFGs) in *operator normal form* —i.e., grammars in which right-hand sides (rhs) of production rules contain no consecutive non-terminals[2]—. As a consequence, the syntax trees generated by such grammars never exhibit two consecutive internal nodes.

The distinguishing feature of OPGs is that they define three *precedence relations* (PR) between pairs of input symbols which drive the deterministic parsing and therefore the construction of a unique ST, if any, associated with an input string. For this reason we consider OPLs a kind of input-driven languages, but larger then the original VPLs. The three PRs are denoted by the symbols $\lessdot, \doteq, \gtrdot$ and are respectively named *yields precedence, equal in precedence*, and *takes precedence*. They graphically resemble the traditional arithmetic relations but do not share their typical ordering and equivalence properties; we kept them for "historical reasons", but we recommend the reader not to be confused by the similarity.

Intuitively, given two input characters a, b belonging to a grammar's *terminal alphabet* separated by at most one non-terminal, $a \lessdot b$ iff in some grammar derivation b is the first terminal character of a grammar's rhs following a whether the grammar's rule contains a non-terminal character before b or not (for this reasons we also say that non-terminal characters are "transparent" in OPL parsing); $a \doteq b$ iff a and b occur consecutively in some rhs, possibly separated by one non-terminal; $a \gtrdot b$ iff a is the last terminal in a rhs —whether followed or not by a non-terminal—, and b follows that rhs in some derivation. The following example provides a first intuition of how a set of *unique* PRs drives the parsing of a string of terminal characters in a deterministic way; subsequently the above concepts are formalized.

Example 2.1. Consider the alphabet of terminal symbols $\Sigma = \{\mathbf{call}, \mathbf{ret}, \mathbf{han}, \mathbf{exc}\}$: as the chosen identifiers suggest, **call** represents the fact that a procedure call occurs, **ret** represents the fact that a procedure terminates normally and returns to its caller, **exc** that an exception is raised and **han** that an exception handler is installed. We want to implement a policy such that an exception aborts all the pending calls up to the point where an appropriate handler is found in the stack, *if any*; after that, execution is resumed normally. Calls and returns, as well as possible pairing of handlers and exceptions are managed according to the usual LIFO policy. The alphabet symbols are written in boldface for reasons that will be explained later but are irrelevant for this example.

The above policy is implemented by the PRs described in Figure 1 which displays the PRs through a square matrix, called *operator precedence matrix (OPM)*, where the element of row i and column j is the PR between the symbol labeling row i and that of column j. We also add the special symbol # which is used as a string delimiter and state the convention that all symbols of Σ yield precedence to, and take precedence over it.

Let us now see how the OPM of Figure 1, named $M_{\mathbf{call}}$, drives the construction of a unique ST associated to a string on the alphabet Σ through a typical bottom-up parsing

[2]Every CFG can be effectively transformed into an equivalent one in operator form [Har78].

$$
\begin{array}{l|l}
0 & \# \lessdot \mathbf{call} \lessdot \mathbf{han} \lessdot \mathbf{call} \lessdot \mathbf{call} \lessdot \underline{\mathbf{call}} \gtrdot \mathbf{exc} \gtrdot \mathbf{call} \doteq \mathbf{ret} \gtrdot \mathbf{call} \doteq \mathbf{ret} \gtrdot \mathbf{ret} \gtrdot \# \\
1 & \# \lessdot \mathbf{call} \lessdot \mathbf{han} \lessdot \mathbf{call} \lessdot \underline{\mathbf{call}}\ N \gtrdot \mathbf{exc} \gtrdot \mathbf{call} \doteq \mathbf{ret} \gtrdot \mathbf{call} \doteq \mathbf{ret} \gtrdot \mathbf{ret} \gtrdot \# \\
2 & \# \lessdot \mathbf{call} \lessdot \mathbf{han} \lessdot \underline{\mathbf{call}}\ N \gtrdot \mathbf{exc} \gtrdot \mathbf{call} \doteq \mathbf{ret} \gtrdot \mathbf{call} \doteq \mathbf{ret} \gtrdot \mathbf{ret} \gtrdot \# \\
3 & \# \lessdot \mathbf{call} \lessdot \underline{\mathbf{han}} \doteq N\ \underline{\mathbf{exc}} \gtrdot \mathbf{call} \doteq \mathbf{ret} \gtrdot \mathbf{call} \doteq \mathbf{ret} \gtrdot \mathbf{ret} \gtrdot \# \\
4 & \# \lessdot \mathbf{call} \lessdot N\ \underline{\mathbf{call}} \doteq \underline{\mathbf{ret}} \gtrdot \mathbf{call} \doteq \mathbf{ret} \gtrdot \mathbf{ret} \gtrdot \# \\
5 & \# \lessdot \mathbf{call} \lessdot N\ \underline{\mathbf{call}} \doteq \underline{\mathbf{ret}} \gtrdot \mathbf{ret} \gtrdot \# \\
6 & \# \lessdot \underline{\mathbf{call}} \doteq N\ \underline{\mathbf{ret}} \gtrdot \# \\
7 & \# \doteq N \#
\end{array}
$$

Figure 2: The sequence of bottom-up reductions during the parsing of w_{ex}.

algorithm. We will see that the shape of the obtained ST depends only on the OPM and not on the particular grammar exhibiting the OPM. Consider the sample word $w_{ex} =$ **call han call call call exc call ret call ret ret**. First, add the delimiter # at its boundaries and write all precedence relations between consecutive characters, according to $M_{\mathbf{call}}$. The result is row 0 of Figure 2.

Then, select all innermost patterns of the form $a \lessdot c_1 \doteq \cdots \doteq c_\ell \gtrdot b$. In row 0 of Figure 2 the only such pattern is the underscored **call** enclosed within the pair $(\lessdot, \gtrdot)$. This means that the ST we are going to build, if it exists, must contain an internal node with the terminal character **call** as its only child. We mark this fact by replacing the pattern $\lessdot\underline{\mathbf{call}}\gtrdot$ with a dummy non-terminal character, say N —i.e., we *reduce* $\underline{\mathbf{call}}$ to N—. The result is row 1 of Figure 2.

Next, we apply the same labeling to row 1 by simply ignoring the presence of the dummy symbol N and we find a new candidate for reduction, namely the pattern $\lessdot\underline{\mathbf{call}}\ N\gtrdot$. Notice that there is no doubt on building the candidate rhs as $\lessdot\underline{\mathbf{call}}\ N\gtrdot$: if we reduced just the $\underline{\mathbf{call}}$ and replaced it by a new N, we would produce two adjacent internal nodes, which is impossible since the ST must be generated by a grammar in operator normal form.

By skipping the obvious reduction of row 2, we come to row 3. This time the terminal characters to be reduced, again, underscored, are two, with an $\doteq$ and an N in between. This means that they embrace a subtree of the whole ST whose root is the node represented by the dummy symbol N. By executing the new reduction leading from row 3 to 4 we produce a new N immediately to the left of a **call** which is matched by an equal in precedence **ret**. Then, the procedure is repeated until the final row 7 is obtained, where, by convention we state the $\doteq$ relation between the two delimiters.

Given that each reduction applied in Figure 2 corresponds to a derivation step of a grammar and to the expansion of an internal node of the corresponding ST, it is immediate to realize that the ST of w_{ex} is the one depicted in Figure 3, where the terminal symbols have been numbered according to their occurrence —including the conventional numbering of the delimiters— for future convenience, and labeling internal nodes has been omitted as useless.

Remarks 2.2. The tree of Figure 3 emphasizes the main difference between various types of parenthesis-like languages, such as VPLs, and OPLs: whereas in the former ones every open parenthesis is consumed by the only corresponding closed one[3], in our example a **call**

[3]To be precise, VPLs allow for unmatched closed parentheses but only at the beginning of a string and unmatched open ones at the end.

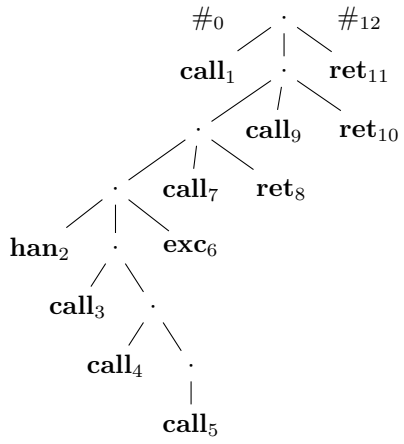

Figure 3: The ST corresponding to word w_{ex}. Dots represent non-terminals.

can be matched by the appropriate **ret** but can also be "aborted" by an **exc** which in turn aborts all pending **call**s until its corresponding **han** —*if any*— is found.

Thus, an OPM defines a *universe* of strings on the given alphabet that can be parsed according to it and assigns a unique ST —with unlabeled internal nodes— to each one of them. Such a universe is the whole Σ^* iff *the OPM is complete*, i.e. it has no empty cells, including those of the implicit row and column referring to the delimiters. In the early literature about OPLs, e.g., [Flo63, CMM78] OPGs sharing a given OPM were used to define restricted languages w.r.t. the universe defined by the OPM and their algebraic properties have been investigated. Later on the same operation has been defined by using different formalisms such as pushdown automata, monadic second order logic, and suitable extensions of regular expressions. In this paper we refer to the use automata and temporal logic, which are typical of model checking. As a side remark we mention that, in general, it may happen that in the same string there are several patterns ready to be reduced; this could enable the implementation of parallel parsing algorithms (see e.g., [BCM+15]) which however is not an issue of interest in this paper.

We now state the basics of OPLs needed for this paper in a formal way. Let Σ be a finite alphabet, and ε the empty string. We use the special symbol $\# \notin \Sigma$ to mark the beginning and the end of any string.

Definition 2.3. An *operator precedence matrix* (OPM) M over Σ is a partial function $(\Sigma \cup \{\#\})^2 \to \{\lessdot, \doteq, \gtrdot\}$, that, for each ordered pair (a, b), defines the *precedence relation* $M(a, b)$ holding between a and b. If the function is total we say that M is *complete*. We call the pair (Σ, M) an *operator precedence alphabet*. By convention, the initial # can only yield precedence to other symbols, and other symbols can only take precedence on the ending #.

If $M(a, b) = \pi$, where $\pi \in \{\lessdot, \doteq, \gtrdot\}$, we write $a\ \pi\ b$. For $u, v \in (\Sigma \cup \{\#\})^+$ we write $u\ \pi\ v$ if $u = xa$ and $v = by$ with $a\ \pi\ b$.

The next concept of *chain* makes the connection between OP relations and ST structure explicit, through brackets.

Definition 2.4. A *simple chain* ${}^{c_0}[c_1 c_2 \dots c_\ell]^{c_{\ell+1}}$, with $\ell \geq 1$, is a string $c_0 c_1 c_2 \dots c_\ell c_{\ell+1}$, such that: $c_0, c_{\ell+1} \in \Sigma \cup \{\#\}$, $c_i \in \Sigma$ for every $i = 1, 2, \dots \ell$, and $c_0 \lessdot c_1 \doteq c_2 \dots c_{\ell-1} \doteq c_\ell \gtrdot c_{\ell+1}$.

A *composed chain* is a string $c_0 s_0 c_1 s_1 c_2 \dots c_\ell s_\ell c_{\ell+1}$, where ${}^{c_0}[c_1 c_2 \dots c_\ell]^{c_{\ell+1}}$ is a simple chain, and $s_i \in \Sigma^*$ is either the empty string or is such that ${}^{c_i}[s_i]^{c_{i+1}}$ is a chain (simple or composed), for every $i = 0, 1, \dots, \ell$ ($\ell \geq 1$). Such a composed chain will be written as ${}^{c_0}[s_0 c_1 s_1 c_2 \dots c_\ell s_\ell]^{c_{\ell+1}}$.

In a chain, simple or composed, c_0 (resp. $c_{\ell+1}$) is called its *left* (resp. *right*) *context*; all terminals between them are called its *body*.

A finite word w over Σ is *compatible* with an OPM M iff for each pair of letters c, d, consecutive in w, $M(c, d)$ is defined and, for each substring x of $\#w\#$ which is a chain of the form ${}^{a}[y]^{b}$, $M(a, b)$ is defined. For a given operator precedence alphabet (Σ, M) the set of all words compatible with M is called the *universe* of the operator precedence alphabet.

The chain below is the chain defined by the OPM $M_{\mathbf{call}}$ of Figure 1 for the word w_{ex}. It shows the natural isomorphism between STs with unlabeled internal nodes (see Figure 3) and chains.

$$\#[\mathbf{call}[[[\mathbf{han}[\mathbf{call}[\mathbf{call}[\mathbf{call}]]]\mathbf{exc}]\mathbf{call\ ret}]\mathbf{call\ ret}]\mathbf{ret}]\#$$

Note that, in composed chains, consecutive inner chains are always separated by an input symbol: this is due to the fact that OPL strings are generated by grammars in operator normal form.

Next we introduce operator precedence automata as pushdown machines suitable to carve specific OPLs within the universe of an operator precedence alphabet.

Definition 2.5. An *operator precedence automaton (OPA)* is a tuple $\mathcal{A} = (\Sigma, M, Q, I, F, \delta)$ where: (Σ, M) is an operator precedence alphabet, Q is a finite set of states (disjoint from Σ), $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states, $\delta \subseteq Q \times (\Sigma \cup Q) \times Q$ is the transition relation, which is the union of the three disjoint relations $\delta_{shift} \subseteq Q \times \Sigma \times Q$, $\delta_{push} \subseteq Q \times \Sigma \times Q$, and $\delta_{pop} \subseteq Q \times Q \times Q$. An OPA is deterministic iff I is a singleton, and all three components of δ are —possibly partial— functions.

To define the semantics of OPA, we need some new notations. Letters $p, q, p_i, q_i, \dots$ denote states in Q. We use $q_0 \xrightarrow{a} q_1$ for $(q_0, a, q_1) \in \delta_{push}$, $q_0 \overset{a}{\dashrightarrow} q_1$ for $(q_0, a, q_1) \in \delta_{shift}$, $q_0 \overset{q_2}{\Longrightarrow} q_1$ for $(q_0, q_2, q_1) \in \delta_{pop}$, and $q_0 \overset{w}{\rightsquigarrow} q_1$, if the automaton can read $w \in \Sigma^*$ going from q_0 to q_1. Let Γ be $\Sigma \times Q$ and $\Gamma' = \Gamma \cup \{\bot\}$ be the *stack alphabet*; we denote symbols in Γ as $[a,\ q]$. We set $smb([a,\ q]) = a$, $smb(\bot) = \#$, and $st([a,\ q]) = q$. For a stack content $\gamma = \gamma_n \dots \gamma_1 \bot$, with $\gamma_i \in \Gamma$, $n \geq 0$, we set $smb(\gamma) = smb(\gamma_n)$ if $n \geq 1$, and $smb(\gamma) = \#$ if $n = 0$.

A *configuration* of an OPA is a triple $c = \langle w,\ q,\ \gamma \rangle$, where $w \in \Sigma^*\#$, $q \in Q$, and $\gamma \in \Gamma^*\bot$. A *computation* or *run* is a finite sequence $c_0 \vdash c_1 \vdash \dots \vdash c_n$ of *moves* or *transitions* $c_i \vdash c_{i+1}$. There are three kinds of moves, depending on the PR between the symbol on top of the stack and the next input symbol:

push move: if $smb(\gamma) \lessdot a$ then $\langle ax,\ p,\ \gamma \rangle \vdash \langle x,\ q,\ [a,\ p]\gamma \rangle$, with $(p, a, q) \in \delta_{push}$;
shift move: if $a \doteq b$ then $\langle bx,\ q,\ [a,\ p]\gamma \rangle \vdash \langle x,\ r,\ [b,\ p]\gamma \rangle$, with $(q, b, r) \in \delta_{shift}$;
pop move: if $a \gtrdot b$ then $\langle bx,\ q,\ [a,\ p]\gamma \rangle \vdash \langle bx,\ r,\ \gamma \rangle$, with $(q, p, r) \in \delta_{pop}$.

Shift and pop moves are not performed when the stack contains only $\bot$. Push moves put a new element on top of the stack consisting of the input symbol together with the current state of the OPA. Shift moves update the top element of the stack by *changing its input symbol only*. Pop moves remove the element on top of the stack, and update the state of the OPA according to δ_{pop} on the basis of the current state and the state

in the removed stack symbol. They do not consume the input symbol, which is used only as a look-ahead to establish the $\gtrdot$ relation. The OPA accepts the language $L(\mathcal{A}) = \{x \in \Sigma^* \mid \langle x\#,\ q_I,\ \bot\rangle \vdash^* \langle \#,\ q_F,\ \bot\rangle, q_I \in I, q_F \in F\}$.

Definition 2.6. Let $\mathcal{A}$ be an OPA. We call a *support* for the simple chain ${}^{c_0}[c_1 c_2 \dots c_\ell]^{c_{\ell+1}}$ any path in $\mathcal{A}$ of the form $q_0 \xrightarrow{c_1} q_1 \dashrightarrow \dots \dashrightarrow q_{\ell-1} \overset{c_\ell}{\dashrightarrow} q_\ell \overset{q_0}{\Longrightarrow} q_{\ell+1}$. The label of the last (and only) pop is exactly q_0, i.e. the first state of the path; this pop is executed because of relation $c_\ell \gtrdot c_{\ell+1}$.

We call a *support for the composed chain* ${}^{c_0}[s_0 c_1 s_1 c_2 \dots c_\ell s_\ell]^{c_{\ell+1}}$ any path in $\mathcal{A}$ of the form $q_0 \overset{s_0}{\rightsquigarrow} q'_0 \xrightarrow{c_1} q_1 \overset{s_1}{\rightsquigarrow} q'_1 \overset{c_2}{\dashrightarrow} \dots \overset{c_\ell}{\dashrightarrow} q_\ell \overset{s_\ell}{\rightsquigarrow} q'_\ell \overset{q'_0}{\Longrightarrow} q_{\ell+1}$ where, for every $i = 0, 1, \dots, \ell$: if $s_i \neq \varepsilon$, then $q_i \overset{s_i}{\rightsquigarrow} q'_i$ is a support for the chain ${}^{c_i}[s_i]^{c_{i+1}}$, else $q'_i = q_i$.

Chains fully determine the parsing structure of any OPA over (Σ, M). If the OPA performs the computation $\langle sb, q_i, [a, q_j]\gamma\rangle \vdash^* \langle b, q_k, \gamma\rangle$, then ${}^a[s]^b$ is necessarily a chain over (Σ, M), and there exists a support like the one above with $s = s_0 c_1 \dots c_\ell s_\ell$ and $q_{\ell+1} = q_k$. This corresponds to the parsing of the string $s_0 c_1 \dots c_\ell s_\ell$ within the context a, b, which contains all information needed to build the subtree whose frontier is that string.

Consider the OPA $\mathcal{A}(\Sigma, M) = (\Sigma, M, \{q\}, \{q\}, \{q\}, \delta_{max})$ where $\delta_{max}(q, q) = q$, and $\delta_{max}(q, c) = q$, $\forall c \in \Sigma$. We call it the *OP Max-Automaton* over (Σ, M). For a max-automaton, each chain has a support; thus, a max-automaton accepts exactly the universe of the operator precedence alphabet. If M is complete, the language accepted by $\mathcal{A}(\Sigma, M)$ is Σ^*. With reference to the OPM $M_{\mathbf{call}}$ of Figure 1, the string **ret call han** is accepted by the max-automaton with structure defined by the chain $\#[[\mathbf{ret}]\mathbf{call}[\mathbf{han}]]\#$.

In conclusion, given an OP alphabet, the OPM M assigns a unique structure to any compatible string in Σ^*; unlike VPLs, such a structure is not visible in the string, and must be built by means of a non-trivial parsing algorithm. An OPA defined on the OP alphabet selects an appropriate subset within the universe of the OP alphabet. OPAs form a Boolean algebra whose universal element is the max-automaton. The language classes recognized by deterministic and non-deterministic OPAs coincide. For a more complete description of the OPL family and of its relations with other CFLs we refer the reader to [MP18].

Example 2.7. For readers not familiar with OPLs, we show how OPAs can naturally model programming languages such as Java and C++. Given a set AP of atomic propositions describing events and states of the program, we use $(\mathcal{P}(AP), M_{AP})$ as the OP alphabet. For convenience, we consider a partitioning of AP into a set of normal propositional labels (in round font), and *structural labels* (in bold). Structural labels define the OP structure of the word: M_{AP} is only defined for subsets of AP containing exactly one structural label, so that given two structural labels $\mathbf{l}_1, \mathbf{l}_2$, for any $a, a', b, b' \in \mathcal{P}(AP)$ s.t. $\mathbf{l}_1 \in a, a'$ and $\mathbf{l}_2 \in b, b'$ we have $M_{AP}(a, b) = M_{AP}(a', b')$. In this way, it is possible to define an OPM on the entire $\mathcal{P}(AP)$ by only giving the relations between structural labels, as we did for $M_{\mathbf{call}}$. Figure 4 shows how to model a procedural program with an OPA. The OPA simulates the program's behavior with respect to the stack, by expressing its execution traces with four event kinds: **call** (resp. **ret**) marks a procedure call (resp. return), **han** the installation of an exception handler by a `try` statement, and **exc** an exception being raised. OPM $M_{\mathbf{call}}$ defines the context-free structure of the word, which is strictly linked with the programming language semantics: the $\lessdot$ PR causes nesting (e.g., **call**s can be nested into other **call**s), and the $\doteq$ PR implies a one-to-one relation, e.g. between a **call** and the **ret** of the same function, and a **han** and the **exc** it catches. Each OPA state represents a line in the source code.

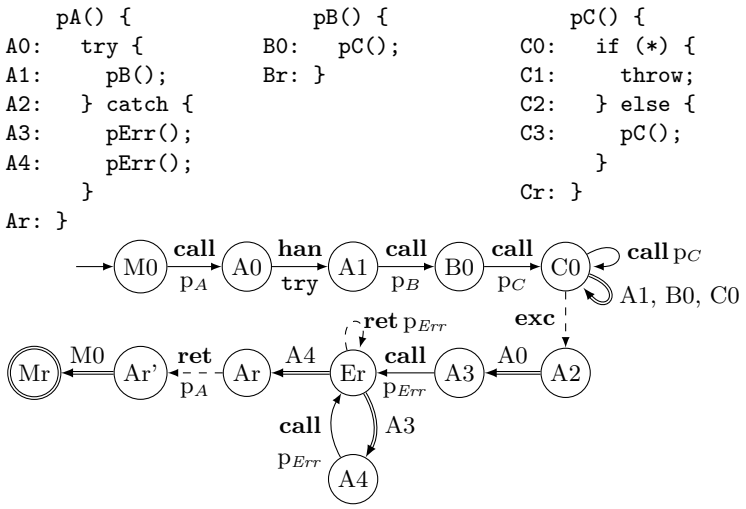

Figure 4: Example procedural program (top) and the derived OPA (bottom). Push, shift, pop moves are shown by, resp., solid, dashed and double arrows.

First, procedure p_A is called by the program loader (M0), and $[\{\mathbf{call}, \mathrm{p}_A\}, \mathrm{M0}]$ is pushed onto the stack, to track the program state before the **call**. Then, the `try` statement at line `A0` of p_A installs a handler. All subsequent calls to p_B and p_C push new stack symbols on top of the one pushed with **han**. p_C may only call itself recursively, or throw an exception, but never return normally. This is reflected by **exc** being the only transition leading from state C0 to the accepting state Mr, and p_B and p_C having no way to a normal **ret**. The OPA has a look-ahead of one input symbol, so when it encounters **exc**, it must pop all symbols in the stack, corresponding to active function frames, until it finds the one with **han** in it, which cannot be popped because $\mathbf{han} \doteq \mathbf{exc}$. Notice that such behavior cannot be modeled by Visibly Pushdown Automata or Nested Word Automata, because they need to read an input symbol for each pop move. Thus, **han** protects the parent function from the exception. Since the state contained in **han**'s stack symbol is A0, the execution resumes in the `catch` clause of p_A. p_A then calls twice the library error-handling function p_{Err}, which ends regularly both times, and returns. The string of Figure 1 is accepted by this OPA.

In this example, we only model the stack behavior for simplicity, but other statements, such as assignments, and other behaviors, such as continuations, could be modeled by a different choice of the OPM, and other aspects of the program's state by appropriate abstractions [JPR18].

2.1. **Operator Precedence ω-Languages.** All definitions regarding OPLs are extended to infinite words in the usual way, but with a few distinctions [LMPP15]. Given an alphabet (Σ, M), an ω-word $w \in \Sigma^\omega$ is compatible with M if every prefix of w is compatible with M. OP ω-words are not terminated by the delimiter #. An ω-word may contain never-ending chains of the form $c_0 \lessdot c_1 \doteq c_2 \doteq \cdots$, where the $\lessdot$ relation between c_0 and c_1 is never closed by a corresponding $\gtrdot$. Such chains are called *open chains* and may be simple or composed. A composed open chain may contain both open and closed subchains. Of course, a closed

chain cannot contain an open one. A terminal symbol $a \in \Sigma$ is *pending* if it is part of the body of an open chain and of no closed chains.

OPA classes accepting the whole class of ωOPLs can be defined by augmenting Definition 2.5 with Büchi or Muller acceptance conditions. In this paper, we only consider the former one. The semantics of configurations, moves and infinite runs are defined as for finite OPAs. For the acceptance condition, let ρ be a run on an ω-word w. Define $\mathrm{Inf}(\rho) = \{q \in Q \mid \text{there exist infinitely many positions } i \text{ s.t. } \langle \beta_i, q, x_i \rangle \in \rho\}$ as the set of states that occur infinitely often in ρ. ρ is successful iff there exists a state $q_f \in F$ such that $q_f \in \mathrm{Inf}(\rho)$. An ωOPBA $\mathcal{A}$ accepts $w \in \Sigma^\omega$ iff there is a successful run of $\mathcal{A}$ on w. The ω-language recognized by $\mathcal{A}$ is $L(\mathcal{A}) = \{w \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } w\}$. Unlike OPAs, ωOPBAs do not require the stack to be empty for word acceptance: when reading an open chain, the stack symbol pushed when the first character of the body of its underlying simple chain is read remains into the stack forever; it is at most updated by shift moves.

The most important closure properties of OPLs are preserved by ωOPLs, which form a Boolean algebra and are closed under concatenation of an OPL with an ωOPL [LMPP15]. The equivalence between deterministic and nondeterministic automata is lost in the infinite case, which is unsurprising, since it also happens for regular ω-languages and ωVPLs.

A more complete treatment of OPLs properties and parsing algorithms can be found in [MP18, GJ08]; ωOPLs are described in some depth in [LMPP15].

2.2. **OPLs vs other structured language families.** The nice closure properties of OPLs come from the fact that a word's syntactic structure is determined locally, once an OPM is given. Other language classes enjoy similar properties. The simplest (and earliest) ones are *Parenthesis Languages* [McN67]. In Parenthesis Languages, two terminals disjoint from the input alphabet are used as open and closed parentheses, and they surround all grammar rule rhs. Thus, the syntactic structure is directly encoded in words (just like we did in Figure 1 with chains).

Visibly Pushdown Languages (VPLs) [AM04], first introduced as Input-Driven Languages [Meh80], extend parenthesis languages; they also lead to applications in model checking. In VPLs, the input alphabet Σ is partitioned into three disjoint sets Σ_c, Σ_i, and Σ_r, called respectively the *call*, *internal*, and *return* alphabets. *Visibly Pushdown Automata* (VPA), the automata class recognizing VPLs, always perform a push move when reading a call symbol, a pop move when reading a return symbol, and, when reading an internal symbol, they perform a move that only updates the current state, leaving the stack untouched. Thus, a string's syntactic structure is fully determined by the alphabet partition, and is clearly *visible*: a symbol in Σ_c is an open parenthesis, and one in Σ_r is a closed parenthesis. The matching between such symbols is unambiguous. Once the alphabet partition is fixed, VPLs form a Boolean algebra, which enabled their success in model checking. VPAs are *real-time*, as they read exactly one input symbol with each move. This limitation distinguishes them from OPAs, whose pop moves are so-called ε-moves. OPLs strictly contain VPLs [CM12].

Nested Words [AM09], an algebraic characterization of VPLs, were introduced to foster their logic and data-theoretic applications. They consist of a linear sequence of positions, plus a *matching relation* encoding the pairing of call and return symbols. As a result, the matching relation is a strictly one-to-one nesting relation that never crosses itself (each "open parenthesis" is matched to a closed one, with a minor exception). The class of Regular Languages of Nested Words is recognized by Nested Words Automata, and is equivalent to VPLs.

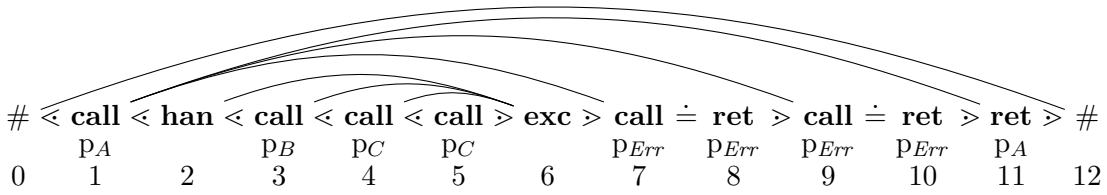

Figure 5: The example string of Figure 1 as an OP word. Chains are highlighted by arrows joining their contexts; structural labels are in bold, and other atomic propositions are shown below them. p_l means a **call** or a **ret** is related to procedure p_l. First, procedure p_A is called (pos. 1), and it installs an exception handler in pos. 2. Then, three nested procedures are called, and the innermost one (p_C) throws an exception, which is caught by the handler. Two more functions are called and, finally, p_A returns.

The original motivation for VPLs was model checking procedural programs: the matching between call and return symbols easily models the behavior of a program's stack during function calls and returns, while internal symbols model other statements. However, many programming languages manage the stack in more complex ways. E.g., when an exception is raised, the stack may be unwound, as multiple procedures are terminated by the exception. In Example 2.7, this is easily modeled by an OPM where the **call** symbol takes precedence from **exc**. In contrast, a VPA would need to read a different symbol for each pop move, so a single **exc** would not suffice.

An early attempt at overcoming such limitations was made with Colored Nested Words [AF16], in which multiple calls can be matched with a return of a different color, allowing many-to-one relations. Colored Nested Words are subsumed by OPLs and, as we show in Section 3, the nesting relation of OPLs can be also one-to-many, besides many-to-one.

3. Precedence-Oriented Temporal Logic

POTL is a linear-time temporal logic, which extends the classical LTL. We recall that the semantics of LTL [Pnu77] is defined on a Dedekind-complete set of word positions U equipped with a total ordering and monadic relations, called *atomic propositions* (AP). In this paper, we consider a discrete timeline, hence $U = \{0, 1, \ldots, n\}$, with $n \in \mathbb{N}$, or $U = \mathbb{N}$. Each LTL formula φ is evaluated in a word position: we write $(w, i) \models \varphi$ to state that φ holds in position i of word w. Besides operators from propositional logic, LTL features modalities that enable movement between positions; e.g., the *Next* modality states that a formula holds in the subsequent position of the current one: $(w, i) \models \bigcirc\varphi$ iff $(w, i+1) \models \varphi$; the *Until* modality states that there exists a *linear path*, made of consecutive positions and starting from the current one, such that a formula ψ holds in the last position of such path, and another formula φ holds in all previous positions. Formally, $(w, i) \models \varphi \,\mathcal{U}\, \psi$ iff there exists $j \geq i$ s.t. $(w, j) \models \psi$, and for all j', with $i \leq j' < j$, we have $(w, j') \models \varphi$.

The linear order, however, is not sufficient to express properties of more complex structures than the linear ones, typically the tree-shaped ones, which are the natural domain of context-free languages. The history of logic formalisms suitable to deal with CFLs

somewhat parallels the path that led from regular languages to tree-languages [Tha67] or their equivalent counterpart in terms of strings, i.e. parenthesis languages [McN67].

A first logic mechanism aimed at "walking through the structure of a context-free sentence" was proposed in [LST94] and consists in a *matching condition* that relates the two extreme terminals of the rhs of a context-free grammar in so-called *double Greibach normal form*, i.e. a grammar whose production rhs exhibit a terminal character at both ends: in a sense such terminal characters play the role of explicit parentheses. [LST94] provides a logic language for general CFLs based on such a relation which however fails to extend the decidability properties of logics for regular languages due to lack of closure properties of CFLs. The matching condition was then resumed in [AM09] to define their MSO logic for VPLs and subsequently the temporal logics CaRet [AEM04] and NWTL [AAB$^+$08].

OPLs are structured but not "visibly structured" as they lack explicit parentheses (see Section 2). Nevertheless, a more sophisticated notion of matching relation has been introduced in [LMPP15] for OPLs by exploiting the fact that OPLs remain input-driven thanks to the OPM. We name the new matching condition *chain relation* and define it here below. We fix a finite set of atomic propositions AP, and an OPM M_{AP} on $\mathcal{P}(AP)$.

A *word structure* —also called *OP word* for short— is the tuple $\langle U, <, M_{AP}, P\rangle$, where U, $<$, and M_{AP} are as above, and $P\colon AP \to \mathcal{P}(U)$ is a function associating each atomic proposition with the set of positions where it holds, with $0, (n+1) \in P(\#)$. For the time being, we consider just finite string languages; the necessary extensions needed to deal with ω-languages will be introduced in Section 3.2.

Definition 3.1 (Chain relation)**.** The *chain relation* $\chi(i, j)$ holds between two positions $i, j \in U$ iff $i < j - 1$, and i and j are resp. the left and right contexts of the same chain (cf. Definition 2.4), according to M_{AP} and the labeling induced by P.

In the following, given two positions i, j and a PR π, we write $i\ \pi\ j$ to say $a\ \pi\ b$, where $a = \{\mathrm{p} \mid i \in P(\mathrm{p})\}$, and $b = \{\mathrm{p} \mid j \in P(\mathrm{p})\}$. For notational convenience, we partition AP into structural labels, written in bold face, which define a word's structure, and normal labels, in round face, defining predicates holding in a position. Thus, an OPM M can be defined on structural labels only, and M_{AP} is obtained by inverse homomorphism of M on subsets of AP containing exactly one of them.

The chain relation augments the linear structure of a word with the tree-like structure of OPLs. Figure 5 shows the word of Figure 3 as an OP word and emphasizes the distinguishing feature of the relation, i.e. that, for composed chains, it may not be one-to-one, but also one-to-many or many-to-one. Notice the correspondence between internal nodes in the ST and pairs of positions in the χ relation.

In the ST, we say that the right context j of a chain is at the *same level* as the left one i when $i \doteq j$ (e.g., in Figure 3, pos. 1 and 11), at a *lower level* when $i \lessdot j$ (e.g., pos. 1 with 7, and 9), at a *higher level* if $i \gtrdot j$ (e.g., pos. 3 and 4 with 6).

Furthermore, given $i, j \in U$, relation χ has the following properties:

(1) It never crosses itself: if $\chi(i, j)$ and $\chi(h, k)$, for any $h, k \in U$, then we have $i < h < j \implies k \leq j$ and $i < k < j \implies i \leq h$.
(2) If $\chi(i, j)$, then $i \lessdot i + 1$ and $j - 1 \gtrdot j$.
(3) Consider all positions (if any) $i_1 < i_2 < \cdots < i_n$ s.t. $\chi(i_p, j)$ for all $1 \leq p \leq n$. We have $i_1 \lessdot j$ or $i_1 \doteq j$ and, if $n > 1$, $i_q \gtrdot j$ for all $2 \leq q \leq n$.
(4) Consider all positions (if any) $j_1 < j_2 < \cdots < j_n$ s.t. $\chi(i, j_p)$ for all $1 \leq p \leq n$. We have $i \gtrdot j_n$ or $i \doteq j_n$ and, if $n > 1$, $i \lessdot j_q$ for all $1 \leq q \leq n - 1$.

Property 4 says that when the chain relation is one-to-many, the contexts of the outermost chain are in the $\doteq$ or $\gtrdot$ relation, while the inner ones are in the $\lessdot$ relation. Property 3 says that contexts of outermost many-to-one chains are in the $\doteq$ or $\lessdot$ relation, and the inner ones are in the $\gtrdot$ relation. Such properties are proved in Appendix A for readers unfamiliar with OPLs.

The χ relation is the core of the MSO logic characterization for OPLs given in [LMPP15] where it is also shown that the greater generality of OPLs and corresponding MSO logic, though requiring more technical proofs, produces results in terms of closure properties, decidability and complexity of the constructions that are the same as the corresponding ones for VPLs. Similarly, in this paper we are going to show that the temporal logic POTL replicates the FO-completeness result of NWTL despite the greater complexity of the χ relation.[4]

While LTL's linear paths only follow the ordering relation $<$, paths in POTL may follow the χ relation too. As a result, a POTL path through a string can simulate paths through the corresponding ST.

We envisage two basic types of path. The first one is that of *summary paths*. By following the chain relation, summary paths may skip chain bodies, which correspond to the fringe of a subtree in the ST. We distinguish between *downward* and *upward* summary paths (resp. DSP and USP). Both kinds can follow both the $<$ and the χ relations; DSPs can enter a chain body but cannot exit it so that they can move only downward in a ST or remain at the same level; conversely, USPs cannot enter one but can move upward by exiting the current one. In other words, if a position k is part of a DSP, and there are two positions i and j, with $i < k < j$ and $\chi(i, j)$ holds, the next position in the DSP cannot be $\geq j$. E.g., two of the DSPs starting from pos. 1 in Figure 5 are 1-2-3, which enters chain $\chi(2, 6)$, and 1-2-6, which skips its body. USPs are symmetric, and some examples thereof are paths 3-6-7 and 4-6-7.

Since the χ relation can be many-to-one or one-to-many, it makes sense to write formulas that consider only left contexts of chains that share their right context, or vice versa. Thus, the paths of our second type, named *hierarchical paths*, are made of such positions, but excluding outermost chains. E.g., in Figure 5, positions 2, 3 and 4 are all in the χ relation with 6, so 3-4 is a hierarchical path ($\chi(2, 6)$ is the outermost chain). Symmetrically, 7-9 is another hierarchical path. The reason for excluding the outermost chain is that, with most OPMs, such positions have a different semantic role than internal ones. E.g., positions 3 and 4 are both calls terminated by the same exception, while 2 is the handler. Positions 7 and 9 are both calls issued by the same function (the one called in position 1), while 11 is its return. This is a consequence of properties 3 and 4 above.

In the next subsection, we describe in a complete and formal way POTL for finite string OPLs while in the subsequent subsection we briefly describe the necessary changes to deal with ω-languages.

3.1. **POTL Syntax and Semantics.** Given a finite set of atomic propositions AP, let $\mathrm{a} \in AP$, and $t \in \{d, u\}$. The syntax of POTL is the following:

$$\begin{aligned}\varphi ::= \mathrm{a} \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc^t\varphi \mid \ominus^t\varphi \mid \chi_F^t\varphi \mid \chi_P^t\varphi \mid \varphi\, \mathcal{U}_\chi^t\, \varphi \mid \varphi\, \mathcal{S}_\chi^t\, \varphi \\ \mid \bigcirc_H^t\varphi \mid \ominus_H^t\varphi \mid \varphi\, \mathcal{U}_H^t\, \varphi \mid \varphi\, \mathcal{S}_H^t\, \varphi\end{aligned}$$

[4] In [CMP21] we produce model checking algorithms with the same order of complexity as those for NWTL.

The truth of POTL formulas is defined w.r.t. a single word position. Let w be an OP word, and a $\in AP$. Then, for any position $i \in U$ of w, we have $(w, i) \models$ a iff $i \in P(\text{a})$. Operators such as $\wedge$ and $\neg$ have the usual semantics from propositional logic. Next, while giving the formal semantics of POTL operators, we illustrate it by showing how it can be used to express properties on program execution traces, such as the one of Figure 5.

Next/back operators. The *downward* next and back operators $\bigcirc^d$ and $\ominus^d$ are like their LTL counterparts, except they are true only if the next (resp. current) position is at a lower or equal ST level than the current (resp. preceding) one. The *upward* next and back, $\bigcirc^u$ and $\ominus^u$, are symmetric. Formally, $(w, i) \models \bigcirc^d\varphi$ iff $(w, i+1) \models \varphi$ and $i \lessdot (i+1)$ or $i \doteq (i+1)$, and $(w, i) \models \ominus^d\varphi$ iff $(w, i-1) \models \varphi$, and $(i-1) \lessdot i$ or $(i-1) \doteq i$. Substitute $\lessdot$ with $\gtrdot$ to obtain the semantics for $\bigcirc^u$ and $\ominus^u$.

E.g., we can write $\bigcirc^d$**call** to say that the next position is an inner call (it holds in pos. 2, 3, 4 of Figure 5), $\ominus^d$**call** to say that the previous position is a **call**, and the current is the first of the body of a function (pos. 2, 4, 5), or the **ret** of an empty one (pos. 8, 10), and $\ominus^u$**call** to say that the current position terminates an empty function frame (holds in 6, 8, 10). In pos. 2 $\bigcirc^d \mathrm{p}_B$ holds, but $\bigcirc^u \mathrm{p}_B$ does not.

Chain Next/Back. The *chain* next and back operators χ_F^t and χ_P^t evaluate their argument resp. on future and past positions in the chain relation with the current one. The *downward* (resp. *upward*) variant only considers chains whose right context goes down (resp. up) or remains at the same level in the ST. Formally, $(w, i) \models \chi_F^d\varphi$ iff there exists a position $j > i$ such that $\chi(i, j)$, $i \lessdot j$ or $i \doteq j$, and $(w, j) \models \varphi$. $(w, i) \models \chi_P^d\varphi$ iff there exists a position $j < i$ such that $\chi(j, i)$, $j \lessdot i$ or $j \doteq i$, and $(w, j) \models \varphi$. Replace $\lessdot$ with $\gtrdot$ for the upward versions.

E.g., in pos. 1 of Figure 5, $\chi_F^d \mathrm{p}_{Err}$ holds because $\chi(1, 7)$ and $\chi(1, 9)$, meaning that p_A calls p_{Err} at least once. Also, χ_F^u**exc** is true in **call** positions whose procedure is terminated by an exception thrown by an inner procedure (e.g. pos. 3 and 4). χ_P^u**call** is true in **exc** statements that terminate at least one procedure other than the one raising it, such as the one in pos. 6. χ_F^d**ret** and χ_F^u**ret** hold in **call**s to non-empty procedures that terminate normally, and not due to an uncaught exception (e.g., pos. 1).

(Summary) Until/Since operators. POTL has two kinds of until and since operators. They express properties on paths, which are sequences of positions obtained by iterating the different kinds of next or back operators. In general, a *path* of length $n \in \mathbb{N}$ between $i, j \in U$ is a sequence of positions $i = i_1 < i_2 < \cdots < i_n = j$. The *until* operator on a set of paths Γ is defined as follows: for any word w and position $i \in U$, and for any two POTL formulas φ and ψ, $(w, i) \models \varphi\,\mathcal{U}(\Gamma)\,\psi$ iff there exist a position $j \in U$, $j \geq i$, and a path $i_1 < i_2 < \cdots < i_n$ between i and j in Γ such that $(w, i_k) \models \varphi$ for any $1 \leq k < n$, and $(w, i_n) \models \psi$. *Since* operators are defined symmetrically. Note that, depending on Γ, a path from i to j may not exist. We define until/since operators by associating them with different sets of paths.

The *summary* until $\psi\,\mathcal{U}_\chi^t\,\theta$ (resp. since $\psi\,\mathcal{S}_\chi^t\,\theta$) operator is obtained by inductively applying the $\bigcirc^t$ and χ_F^t (resp. $\ominus^t$ and χ_P^t) operators. It holds in a position in which either θ holds, or ψ holds together with $\bigcirc^t(\psi\mathcal{U}_\chi^t\,\theta)$ (resp. $\ominus^t(\psi\,\mathcal{S}_\chi^t\,\theta)$) or $\chi_F^t(\psi\mathcal{U}_\chi^t\,\theta)$ (resp. $\chi_P^t(\psi\,\mathcal{S}_\chi^t\,\theta)$). It is an until operator on paths that can move not only between consecutive positions, but also between contexts of a chain, skipping its body. With the OPM of Figure 1, this means skipping function bodies. The downward variants can move between positions at the same

level in the ST (i.e., in the same simple chain body), or down in the nested chain structure. The upward ones remain at the same level, or move to higher levels of the ST.

Formula $\top\,\mathcal{U}^u_\chi\,\mathbf{exc}$ is true in positions contained in the frame of a function that is terminated by an exception. It is true in pos. 3 of Figure 5 because of path 3-6, and false in pos. 1, because no upward path can enter the chain whose contexts are pos. 1 and 11. Formula $\top\,\mathcal{U}^d_\chi\,\mathbf{exc}$ is true in call positions whose function frame contains **exc**s, but that are not directly terminated by one of them, such as the one in pos. 1 (with path 1-2-6).

We formally define *Downward Summary Paths* (DSP) as follows. Given an OP word w, and two positions $i \leq j$ in w, the DSP between i and j, if it exists, is a sequence of positions $i = i_1 < i_2 < \cdots < i_n = j$ such that, for each $1 \leq p < n$,

$$i_{p+1} = \begin{cases} k & \text{if } k = \max\{h \mid h \leq j \wedge \chi(i_p, h) \wedge (i_p \lessdot h \vee i_p \doteq h)\} \text{ exists;} \\ i_p + 1 & \text{otherwise, if } i_p \lessdot (i_p + 1) \text{ or } i_p \doteq (i_p + 1). \end{cases}$$

The Downward Summary (DS) until and since operators $\mathcal{U}^d_\chi$ and $\mathcal{S}^d_\chi$ use as Γ the set of DSPs starting in the position in which they are evaluated. The definition for the upward counterparts is, again, obtained by substituting $\gtrdot$ for $\lessdot$. In Figure 5, $\mathbf{call}\,\mathcal{U}^d_\chi\,(\mathbf{ret} \wedge \mathrm{p}_{Err})$ holds in pos. 1 because of path 1-7-8 and 1-9-10, $(\mathbf{call} \vee \mathbf{exc})\,\mathcal{S}^u_\chi\,\mathrm{p}_B$ in pos. 7 because of path 3-6-7, and $(\mathbf{call} \vee \mathbf{exc})\,\mathcal{U}^u_\chi\,\mathbf{ret}$ in 3 because of path 3-6-7-8.

Hierarchical operators. A single position may be the left or right context of multiple chains. The operators seen so far cannot keep this fact into account, since they "forget" about a left context when they jump to the right one. Thus, we introduce the *hierarchical* next and back operators. The *upward* hierarchical next (resp. back), $\bigcirc^u_H\psi$ (resp. $\ominus^u_H\psi$), is true iff the current position j is the right context of a chain whose left context is i, and ψ holds in the next (resp. previous) pos. j' that is a right context of i, with $i \lessdot j, j'$. So, $\bigcirc^u_H\mathrm{p}_{Err}$ holds in pos. 7 of Figure 5 because p_{Err} holds in 9, and $\ominus^u_H\mathrm{p}_{Err}$ in 9 because p_{Err} holds in 7. In the ST, $\bigcirc^u_H$ goes *up* between **call**s to p_{Err}, while $\ominus^u_H$ goes down. Their *downward* counterparts behave symmetrically, and consider multiple inner chains sharing their right context. They are formally defined as:

- $(w,i) \models \bigcirc^u_H\varphi$ iff there exist a position $h < i$ s.t. $\chi(h,i)$ and $h \lessdot i$ and a position $j = \min\{k \mid i < k \wedge \chi(h,k) \wedge h \lessdot k\}$ and $(w,j) \models \varphi$;
- $(w,i) \models \ominus^u_H\varphi$ iff there exist a position $h < i$ s.t. $\chi(h,i)$ and $h \lessdot i$ and a position $j = \max\{k \mid k < i \wedge \chi(h,k) \wedge h \lessdot k\}$ and $(w,j) \models \varphi$;
- $(w,i) \models \bigcirc^d_H\varphi$ iff there exist a position $h > i$ s.t. $\chi(i,h)$ and $i \gtrdot h$ and a position $j = \min\{k \mid i < k \wedge \chi(k,h) \wedge k \gtrdot h\}$ and $(w,j) \models \varphi$;
- $(w,i) \models \ominus^d_H\varphi$ iff there exist a position $h > i$ s.t. $\chi(i,h)$ and $i \gtrdot h$ and a position $j = \max\{k \mid k < i \wedge \chi(k,h) \wedge k \gtrdot h\}$ and $(w,j) \models \varphi$.

In the ST of Figure 3, $\bigcirc^d_H$ and $\ominus^d_H$ go *down* and up among **call**s terminated by the same **exc**. For example, in pos. 3 $\bigcirc^d_H\mathrm{p}_C$ holds, because both pos. 3 and 4 are in the chain relation with 6. Similarly, in pos. 4 $\ominus^d_H\mathrm{p}_B$ holds. Note that these operators do not consider leftmost/rightmost contexts, so $\bigcirc^u_H\mathbf{ret}$ is false in pos. 9, as $\mathbf{call} \doteq \mathbf{ret}$, and pos. 11 is the rightmost context of pos. 1.

The hierarchical until and since operators are defined by iterating these next and back operators. The upward hierarchical path (UHP) between i and j is a sequence of positions $i = i_1 < i_2 < \cdots < i_n = j$ such that there exists a position $h < i$ such that for each $1 \leq p \leq n$ we have $\chi(h, i_p)$ and $h \lessdot i_p$, and for each $1 \leq q < n$ there exists no position k

such that $i_q < k < i_{q+1}$ and $\chi(h, k)$. The until and since operators based on the set of UHPs starting in the position in which they are evaluated are denoted as $\mathcal{U}_H^u$ and $\mathcal{S}_H^u$. E.g., $\mathbf{call}\,\mathcal{U}_H^u\ \mathrm{p}_{Err}$ holds in pos. 7 because of the singleton path 7 and path 7-9, and $\mathbf{call}\,\mathcal{S}_H^u\ \mathrm{p}_{Err}$ in pos. 9 because of paths 9 and 7-9.

The downward hierarchical path (DHP) between i and j is a sequence of positions $i = i_1 < i_2 < \cdots < i_n = j$ such that there exists a position $h > j$ such that for each $1 \leq p \leq n$ we have $\chi(i_p, h)$ and $i_p \gtrdot h$, and for each $1 \leq q < n$ there exists no position k such that $i_q < k < i_{q+1}$ and $\chi(k, h)$. The until and since operators based on the set of DHPs starting in the position in which they are evaluated are denoted as $\mathcal{U}_H^d$ and $\mathcal{S}_H^d$. In Figure 5, $\mathbf{call}\,\mathcal{U}_H^d\ \mathrm{p}_C$ holds in pos. 3, and $\mathbf{call}\,\mathcal{S}_H^d\ \mathrm{p}_B$ in pos. 4, both because of path 3-4.

Equivalences. The POTL until and since operators enjoy expansion laws similar to those of LTL. Here we give those for two until operators, those for their since and downward counterparts being symmetric. All such laws are proved in Appendix B.

$$\varphi\,\mathcal{U}_\chi^t\,\psi \equiv \psi \vee \Big(\varphi \wedge \big(\bigcirc^t(\varphi\,\mathcal{U}_\chi^t\,\psi) \vee \chi_F^t(\varphi\,\mathcal{U}_\chi^t\,\psi)\big)\Big)$$
$$\varphi\,\mathcal{U}_H^u\,\psi \equiv (\psi \wedge \chi_P^d\top \wedge \neg\chi_P^u\top) \vee \big(\varphi \wedge \bigcirc_H^u(\varphi\,\mathcal{U}_H^u\,\psi)\big)$$

As in LTL, it is worth defining some derived operators. For $t \in \{d, u\}$, we define the downward/upward summary *eventually* as $\Diamond^t\varphi := \top\,\mathcal{U}_\chi^t\,\varphi$, and the downward/upward summary *globally* as $\Box^t\varphi := \neg\Diamond^t(\neg\varphi)$. $\Diamond^u\varphi$ and $\Box^u\varphi$ respectively say that φ holds in one or all positions in the path from the current position to the root of the ST. Their downward counterparts are more interesting: they consider all positions in the current rhs and its subtrees, starting from the current position. $\Diamond^d\varphi$ says that φ holds in at least one of such positions, and $\Box^d\varphi$ in all of them. E.g., if $\Box^d(\neg\mathrm{p}_A)$ holds in a **call**, it means that p_A never holds in its whole function body, which is the subtree rooted next to the **call**.

We anticipate that preventing downward paths from crossing the boundaries of the current subtrees and conversely imposing upward ones to exit it without entering any inner one adds, rather than limiting, generality w.r.t. paths that can cross both such boundaries.

3.2. **POTL on ω-Words.** Since applications in model checking usually require temporal logics on infinite words, we now extend POTL to ω-words.

To define OP ω-words, it suffices to replace the finite set of positions U with the set of natural numbers $\mathbb{N}$ in the definition of OP words. Then, the formal semantics of all POTL operators remains the same as in Section 3.1. The only difference in the intuitive meaning of operators occurs in ω-words with open chains. In fact, chain next operators (χ_F^d and χ_F^u) do not hold on the left contexts of open chains, as the χ relation is undefined on them. The same can be said for downward hierarchical operators, when evaluated on left contexts of open chains.

Also, property 4 of the χ relation does not hold if a position i is the left context of an open chain. In this case, there may be positions $j_1 < j_2 < \cdots < j_n$ s.t. $\chi(i, j_p)$ and $i \lessdot j_p$ for all $1 \leq p \leq n$, but no position k s.t. $\chi(i, k)$ and $i \gtrdot k$ or $i \doteq k$.

3.3. **Motivating Examples.** POTL can express many useful requirements of procedural programs. To emphasize the potential practical applications in automatic verification, we supply a few examples of typical program properties expressed as POTL formulas.

Let $\Box\psi$ be the LTL *globally* operator, which can be expressed in POTL as in Section 3.4.1. POTL can express Hoare-style pre/postconditions with formulas such as $\Box(\mathbf{call} \wedge \rho \implies \chi^d_F(\mathbf{ret} \wedge \theta))$, where ρ is the precondition, and θ is the postcondition.

Unlike NWTL, POTL can easily express properties related to exception handling and interrupt management. E.g., the shortcut $\mathit{CallThr}(\psi) := \bigcirc^u(\mathbf{exc} \wedge \psi) \vee \chi^u_F(\mathbf{exc} \wedge \psi)$, evaluated in a **call**, states that the procedure currently started is terminated by an **exc** in which ψ holds. So, $\Box(\mathbf{call} \wedge \rho \wedge \mathit{CallThr}(\top) \implies \mathit{CallThr}(\theta))$ means that if precondition ρ holds when a procedure is called, then postcondition θ must hold if that procedure is terminated by an exception. In object oriented programming languages, if $\rho \equiv \theta$ is a class invariant asserting that a class instance's state is valid, this formula expresses *weak (or basic) exception safety* [Abr00], and *strong exception safety* if ρ and θ express particular states of the class instance. The *no-throw guarantee* can be stated with $\Box(\mathbf{call} \wedge \mathrm{p}_A \implies \neg\mathit{CallThr}(\top))$, meaning procedure p_A is never interrupted by an exception.

Stack inspection [EKS03, JLT99], i.e. properties regarding the sequence of procedures active in the program's stack at a certain point of its execution, is an important class of requirements that can be expressed with shortcut $\mathit{Scall}(\varphi, \psi) := (\mathbf{call} \implies \varphi)\, \mathcal{S}^d_\chi\, (\mathbf{call} \wedge \psi)$, which subsumes the *call since* of CaRet, and works with exceptions too. E.g., $\Box\big((\mathbf{call} \wedge \mathrm{p}_B \wedge \mathit{Scall}(\top, \mathrm{p}_A)) \implies \mathit{CallThr}(\top)\big)$ means that whenever p_B is executed and at least one instance of p_A is on the stack, p_B is terminated by an exception. The OPA of Figure 4 satisfies this formula, because p_B is always called by p_A, and p_C always throws. If the OPA was an ωOPBA, it would not satisfy such formula because of computations where p_C does not terminate.

3.4. **Comparison with the state of the art.**

3.4.1. *Linear Temporal Logic (LTL).* The main limitation of LTL is that the algebraic structure it is defined on only contains a linear order on word positions. Thus, it fails to model systems that require an additional binary relation, such as the χ relation of POTL. LTL is in fact expressively equivalent to the first-order fragment of regular languages, and it cannot represent context-free languages, as POTL does. LTL model checking on pushdown formalisms has been investigated extensively [EHRS00, ABE+05].

On the other hand, POTL can express all LTL operators, so that POTL is strictly more expressive than LTL. Any LTL Next formula $\bigcirc\varphi$ is in fact equivalent to the POTL formula $\bigcirc^d\varphi' \vee \bigcirc^u\varphi'$, where φ' is the translation of φ into POTL, and the LTL Back can be translated symmetrically.

The *Globally* operator can be translated as $\Box\psi := \neg\Diamond^u(\Diamond^d\neg\psi)$. This formula contains an upward summary eventually followed by a downward one, and it can be explained by thinking to a word's ST. The upward eventually evaluates its argument on all positions from the current one to the root. Its argument considers paths from each one of such positions to all the leaves of the subtrees rooted at their right, in the same rhs. Together, the two eventually operators consider paths from the current position to all subsequent positions in the word. Thus, with the initial negation this formula means that $\neg\psi$ never holds in such positions, which is the meaning of the LTL Globally operator.

The translation for LTL Until and Since is much more involved. We need to define some shortcuts, that will be used again in Section 4.2.2. For any $a \subseteq AP$, $\sigma_a := \bigwedge_{\mathrm{p} \in a} \mathrm{p} \wedge \bigwedge_{\mathrm{q} \notin a} \neg \mathrm{q}$ holds in a position i iff a is the set of atomic propositions holding in i. For any POTL formula γ, let $\chi_F^{\lessdot}\gamma := \bigvee_{a,b \subseteq AP,\, a \lessdot b} (\sigma_a \wedge \chi_F^d(\sigma_b \wedge \gamma))$ be the restriction of $\chi_F^d\gamma$ to chains with contexts in the $\lessdot$ PR; $\chi_P^{\gtrdot}\gamma$ is analogous.

The translation for $\varphi \,\mathcal{U}\, \psi$ follows, that for LTL Since being symmetric:

$$\psi' \vee \big(\varphi' \wedge \alpha(\varphi')\big)\, \mathcal{U}_\chi^u\, \big(\psi' \vee (\varphi' \wedge \beta(\varphi'))\, \mathcal{U}_\chi^d\, (\psi' \wedge \beta(\varphi'))\big)$$

where φ' and ψ' are the translations of φ and ψ into POTL, and

$$\alpha(\varphi') := \chi_F^u \top \implies \neg\big(\bigcirc^d (\top\, \mathcal{U}_\chi^d\, \neg\varphi') \vee \chi_F^{\lessdot}(\top\, \mathcal{U}_\chi^d\, \neg\varphi')\big)$$
$$\beta(\varphi') := \chi_P^d \top \implies \neg\big(\ominus^u (\top\, \mathcal{S}_\chi^u\, \neg\varphi') \vee \chi_P^{\gtrdot}(\top\, \mathcal{S}_\chi^u\, \neg\varphi')\big)$$

The main formula is the concatenation of a US until and a DS until, and it can be explained similarly to the translation for LTL Globally. Let i be the word position in which the formula is evaluated, and j the last one of the linear path, in which ψ' holds. The outermost US until is witnessed by a path from i to a position i' which, in the ST, is part of the rhs which is the closest common ancestor of i and j. In all positions $i < k < i'$ in this path, formula $\alpha(\varphi)$ holds. It means φ' holds in all positions contained in the subtree rooted at the non-terminal to the immediate right of k (i.e., in the body of the chain whose left context is k).

Then, the DS until is witnessed by a downward path from i' to j. Here $\beta(\varphi)$ has a role symmetric to $\alpha(\varphi)$. It forces φ' to hold in all subtrees rooted at the non-terminal to the left of positions in the DS path.

Thus, formulas $\alpha(\varphi)$ and $\beta(\varphi)$ make sure φ' holds in all chain bodies skipped by the summary paths, and ψ' holds in j.

3.4.2. *Logics on Nested Words.* The first temporal logics with explicit context-free aware modalities were based on nested words (cf. Section 2.2).

CaRet [AEM04] was the first temporal logic on nested words to be introduced, and it focuses on expressing properties on procedural programs, which explains its choice of modalities. The *Abstract* Next and Until operators are defined on paths of positions in the frame of the same function, skipping frames of nested calls. The *Caller* Next and Until are actually past modalities, and they operate on paths made of the calls of function frames containing the current position. LTL Next and Until are also present. The Caller operators enable upward movement in the ST of a nested word, and abstract operators enable movement in the same rhs. However, no CaRet operator allows pure downward movement in the ST, which is needed to express properties limited to a single subtree. While the LTL until can go downward, it can also go beyond the rightmost leaf of a subtree, thus effectively jumping upwards.

This seems to be the main expressive limitation of CaRet, which is conjectured not to be FO-complete [AAB$^+$08]. In fact, FO-complete temporal logics were introduced in [AAB$^+$08] by adding various kinds of *Within* modalities to CaRet. Such operators limit their operands to span only positions within the same call-return pair, and hence the same subtree of the ST, at the cost of an exponential jump in the complexity of model checking.

Another approach to FO-completeness is that of NWTL [AAB$^+$08], which is based on *Summary* Until and Since operators. Summary paths are made of either consecutive positions, or matched call-return pairs. Thus, they can skip function bodies, and enter or

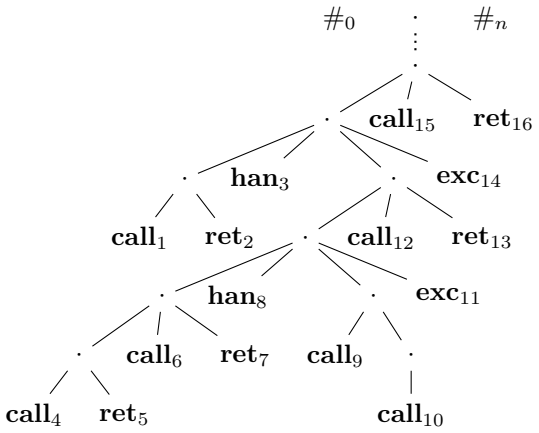

Figure 6: Example OP word on OPM $M_{\mathbf{call}}$.

exit them. Summary-up and down paths, and the respective operators, can be obtained from summary paths, enabling exclusive upward or downward movement in the ST. In particular, summary-down operators may express properties limited to a single subtree.

In Corollary 4.11, we show that CaRet $\subseteq$ NWTL $\subset$ POTL.

3.4.3. *Logics on OPLs.* The only way to overcome the limitations of nested words is to base a temporal logic on a more general algebraic structure. OPTL [CMP20] was introduced with this aim, but it shares some of the limitations that CaRet has on nested words. It features all LTL past and future operators, plus the *Matching* Next ($\bigcirc_\chi$) and Back ($\ominus_\chi$) operators, resp. equivalent to POTL χ_F^u and χ_P^u, OP *Summary* Until and Since, and *Hierarchical* Until and Since. POTL has several advantages over OPTL in its ease of expressing requirements.

Given a set of PRs Π, an OPTL Summary Until $\mathcal{U}^\Pi$ evaluated on a position i considers paths $i = i_1 < i_2 < \cdots < i_n = j$, with $i \leq j$, such that for any $1 \leq p < n$,

$$i_{p+1} = \begin{cases} h & \text{if there exists } h \text{ s.t. } \chi(i_p, h) \text{ and } (i_p \doteq h \text{ or } i_p \gtrdot h), \text{ and } h \leq j; \\ i_p + 1 & \text{if } i_p \; \pi \; i_{p+1} \text{ for some } \pi \in \Pi, \text{ otherwise.} \end{cases}$$

The Summary Since $\mathcal{S}^\Pi$ is symmetric: positions in the χ relation must be in the $\lessdot$ or $\doteq$ PRs. Since the PRs checked on chain contexts are fixed, the user can control whether such paths go up or down in the ST only partially. OPTL's $\varphi \, \mathcal{U}^{\doteq \gtrdot} \, \psi$ is equivalent to POTL's $\varphi \, \mathcal{U}_\chi^u \, \psi$, and $\varphi \, \mathcal{S}^{\lessdot \doteq} \, \psi$ to $\varphi \, \mathcal{S}_\chi^d \, \psi$: both operators only go upwards in the ST. However, there is no OPTL operator equivalent to POTL's $\mathcal{U}_\chi^d$ or $\mathcal{S}_\chi^u$, which go downward. This makes it difficult to express function-local properties limited to a single subtree in OPTL.

E.g., consider POTL formula $\alpha := \Box(\mathbf{exc} \implies \chi_P^d(\mathbf{han} \wedge \Diamond^d \mathrm{p}_A))$, which means that if an exception is thrown, it is always caught, and procedure p_A is called at some point inside the **han-exc** block. One could try to translate it into OPTL with a formula such as $\beta := \Box(\mathbf{exc} \implies \ominus_\chi(\mathbf{han} \wedge \top \, \mathcal{U}^{\lessdot \doteq} \, \mathrm{p}_A))$. Consider the OP word of Figure 6. When the until in β is evaluated in the **han** of pos. 3, its paths can only consider positions 4 and 5, because paths touching such positions cannot pass the $\gtrdot$ relation between 5 and 6. Its paths, unlike those of its POTL counterpart, cannot jump between chain contexts in the $\lessdot$ relation, and cannot reach positions 6, 8, and 12 in this way. If p_A held in pos. 6 and 10, α would be

true in the word, but β would be false. Replacing the until with $\top\, \mathcal{U}^{\lessdot \doteq \gtrdot}\, \mathrm{p}_A$ overcomes such issues, but it introduces another one: its paths would go past pos. 14, going outside of the subtree. Thus, if p_A held only in pos. 15, α would be false, but this variant of β would hold.

OPTL has Hierarchical operators, too. Its yield-precedence Hierarchical Until ($\mathcal{U}^{\uparrow}$) and since ($\mathcal{S}^{\downarrow}$) operators, when evaluated on a position i, consider paths $i < j_1 < j_2 < \cdots < j_n$ s.t. $\chi(i, j_p)$ and $i \lessdot j_p$ for all $1 \leq p \leq n$, and there is no $k < j_1$ s.t. $\chi(i, k)$. Their take-precedence counterparts ($\mathcal{U}^{\downarrow}$ and $\mathcal{S}^{\uparrow}$) are symmetric. Thus, such paths do not start in the position where until and since operators are evaluated, but always in a future position: this is another limitation of OPTL. In fact, it is not possible to concatenate them to express complex properties on right (resp. left) contexts of chains sharing their left (resp. right) context, such as several function calls issued by the same function, or multiple function calls terminated by the same exception. POTL has both Hierarchical Next/Back and Until/Since pairs which are composable, making it expressively complete on such positions. For example, POTL formula $\gamma := \square(\mathbf{call} \wedge \mathrm{p}_A \implies (\top\, \mathcal{S}^u_H\, (\mathbf{call} \wedge \mathrm{p}_B))\, \mathcal{S}^d_\chi\, (\ominus^d \# \vee \chi^d_P \#))$ means that whenever procedure p_A is called, all procedures in the stack have previously invoked p_B (possibly excluding the one directly calling p_A). While $\mathcal{S}^d_\chi$ can be replaced with OPTL's $\mathcal{S}^{\lessdot \doteq}$, POTL's $\mathcal{S}^u_H$ cannot be easily translated. In fact, OPTL's Hierarchical operators would only allow us to state that p_B is invoked by the procedures in the stack, but not necessarily before the call to p_A.

The above intuition about OPTL's weaknesses are made formal in the next subsection.

3.4.4. *OPTL is not expressively complete.* In this section, we prove that no OPTL formula is equivalent to POTL formula $\Diamond^d \mathrm{p}_A$. The proof is quite elaborate, which is unsurprising, since the analogous problem of the comparison between CaRet and NWTL is still open.

First, we prove the following

Lemma 3.2 (Pumping Lemma for OPTL)**.** *Let φ be an OPTL formula and L an OPL, both defined on a set of atomic propositions AP and an OPM M_{AP}. Then, for some positive integer n, for each $w \in L$, $|w| \geq n$, there exist strings $u, v, x, y, z \in \mathcal{P}(AP)^*$ such that $w = uvxyz$, $|vy| > 1$, $|vxy| \leq n$ and for any $k > 0$ we have $w' = uv^k xy^k z \in L$; for any $0 \leq j \leq k$ and $0 \leq i < |v|$ we have $(w, |u| + i) \models \varphi$ iff $(w', |u| + j|v| + i) \models \varphi$, and for any $0 \leq i < |y|$ we have $(w, |uv^k x| + i) \models \varphi$ iff $(w', |uv^k x| + j|y| + i) \models \varphi$.*

Proof. Given a word $w \in L$, we define $\lambda(w)$ as the word of length $|w|$ such that, if position i of w is labeled with a, then the same position in $\lambda(w)$ is labeled with $(a, 1)$ if $(w, i) \models \varphi$, and with $(a, 0)$ otherwise. Let $\lambda(L) = \{\lambda(w) \mid w \in L\}$, and λ^{-1} is such that $\lambda^{-1}(\lambda(w)) = w$. If we prove that $\lambda(L)$ is context-free, from the classic Pumping Lemma [Har78] follows that, for some $n > 0$, for all $\hat{w} \in \lambda(L)$ there exist strings $\hat{u}, \hat{v}, \hat{x}, \hat{y}, \hat{z} \in (\mathcal{P}(AP) \times \{0, 1\})^*$ such that $\hat{w} = \hat{u}\hat{v}\hat{x}\hat{y}\hat{z}$, $|\hat{v}\hat{y}| > 1$, $|\hat{v}\hat{x}\hat{y}| \leq n$ and for any $k > 0$ we have $\hat{w}' = \hat{u}\hat{v}^k\hat{x}\hat{y}^k\hat{z} \in \lambda(L)$. The claim follows by applying λ^{-1} to such strings, and the word positions in which φ holds in $\lambda^{-1}(\hat{w})$ are those labeled with 1.

To prove that $\lambda(L)$ is context-free, we use the OPTL model checking construction given in [CMP20], which yields an OPA $\mathcal{A}_\varphi = (\mathcal{P}(AP), M_{AP}, Q, I, F, \delta)$ accepting models of φ. The states of $\mathcal{A}_\varphi$ are elements of the set $\mathrm{Cl}(\varphi)$, which contains φ and all its subformulas. Given a word w compatible with M_{AP}, the accepting computations of $\mathcal{A}_\varphi$ are such that, for each $0 \leq i < |w|$, the state of $\mathcal{A}_\varphi$ prior to reading position i contains $\psi \in \mathrm{Cl}(\varphi)$ iff $(w, i) \models \psi$.

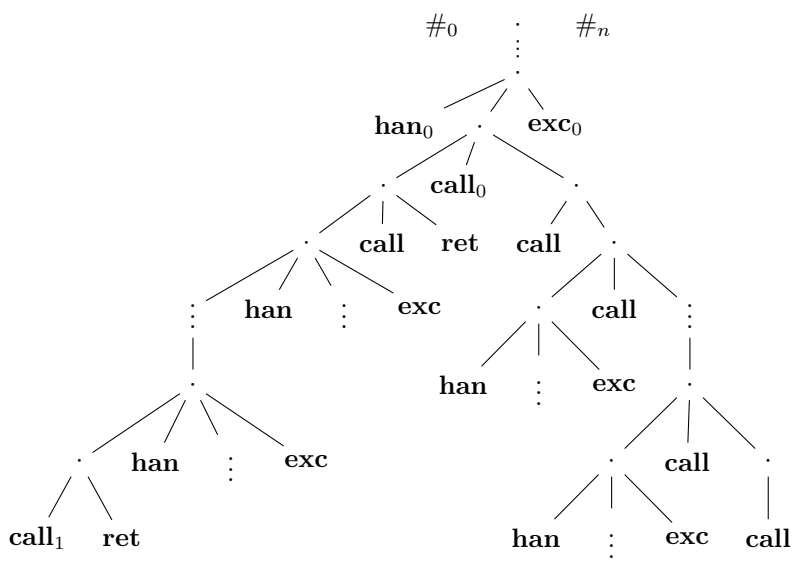

Figure 7: Structure of a word in $L_{\mathbf{call}}$.

Thus, we build OPA $\mathcal{A}_{\lambda(\mathcal{P}(AP)^*)} = (\mathcal{P}(AP) \times \{0,1\}, M_{AP}, Q, I', F, \delta')$ that reads words on $(\mathcal{P}(AP) \times \{0,1\})^*$ and accepts $\lambda(\mathcal{P}(AP)^*)$ as follows:

- I' is the set of all states in Q not containing past operators (and possibly φ);
- δ'_{push} is such that if $(\Phi, a, \Theta) \in \delta_{push}$, then $(\Phi, (a,1), \Theta) \in \delta_{push}$ if $\varphi \in \Phi$, and $(\Phi, (a,0), \Theta) \in \delta_{push}$ otherwise;
- δ'_{shift} is derived from δ_{shift} similarly;
- $\delta'_{pop} = \delta_{pop}$.

Since L is an OPL, there exists an OPA $\mathcal{A}_L$ accepting it. $\mathcal{A}_L$ can be easily modified to obtain $\mathcal{A}'_L$, an OPA accepting all words $\hat{w} \in (\mathcal{P}(AP) \times \{0,1\})^*$ such that the underlying word $w \in \mathcal{P}(AP)^*$ is in L. Language $\lambda(L)$ is the intersection between the language accepted by $\mathcal{A}'_L$, and $\lambda(\mathcal{P}(AP)^*)$. Since OPLs are closed under intersection, $\lambda(L)$ is also an OPL. □

Let $L_{\mathbf{call}}$ be the max-language generated by OPM $M_{\mathbf{call}}$, with the addition that p_A may appear in any word position. We prove the following:

Theorem 3.3. *Given the POTL formula $\Diamond^d \mathrm{p}_A$, for every OPTL formula φ there exist a word $w \in L_{\mathbf{call}}$ and an integer $0 \leq i < |w|$ such that either $(w,i) \models \Diamond^d \mathrm{p}_A$ and $(w,i) \not\models \varphi$, or $(w,i) \not\models \Diamond^d \mathrm{p}_A$ and $(w,i) \models \varphi$.*

Proof. Figure 7 shows the structure of the syntax trees of words in a subset of $L_{\mathbf{call}}$. Dots between **han**–**exc** pairs can be replaced with repetitions of the whole tree structure, and other dots with the repetition of surrounding tree fragments (e.g., **call**–**ret** or **han**–**exc**). $\mathbf{han}_0$ is the position in which φ is evaluated, and $\mathbf{exc}_0$ is its matched **exc**. $\mathbf{call}_1$ is the **call** right after $\mathbf{han}_0$, and $\mathbf{call}_0$ is the one at the highest level of the subtree between $\mathbf{han}_0$ and $\mathbf{exc}_0$. **call**s between $\mathbf{call}_0$ and $\mathbf{exc}_0$ do not have a corresponding **ret**, and are terminated by $\mathbf{exc}_0$. The word delimited by $\mathbf{han}_0$ and $\mathbf{exc}_0$ is itself part of a larger tree with the same structure. For φ to be equivalent to $\Diamond^d \mathrm{p}_A$, it must be able to

(1) look for the symbol p_A in all positions between $\mathbf{han}_0$ and $\mathbf{exc}_0$; and

(2) not consider any positions before $\mathbf{han}_0$ or after $\mathbf{exc}_0$.

In the following, we show that any OPTL formula φ fails to satisfy both requirements, thus

(1) one can hide p_A in one of the positions not covered, so that φ is false in $\mathbf{han}_0$, but $\Diamond^d \mathrm{p}_A$ is true; or
(2) put p_A in one of the positions outside $\mathbf{han}_0$–$\mathbf{exc}_0$ reached by φ, so that it is true in $\mathbf{han}_0$, but $\Diamond^d \mathrm{p}_A$ is not.

If φ is evaluated on position $\mathbf{han}_0$, it must contain some modal operator based on paths that reach each position between $\mathbf{han}_0$ and $\mathbf{exc}_0$. The length of the word between $\mathbf{han}_0$ and $\mathbf{exc}_0$ has no limit, so φ must contain at least an until operator, which may be a LTL Until, an OPTL Hierarchical Until, or an OPTL Summary Until $\mathcal{U}^\Pi$. For $\mathcal{U}^\Pi$, we must have $\succcurlyeq \in \Pi$, or the path would not be able to reach past position $\mathbf{ret}_1$ (remember that an OPTL summary path cannot skip chains with contexts in the $\lessdot$ relation). The presence of $\succcurlyeq$ allows the Summary Until to reach positions past $\mathbf{exc}_0$: the formula could be true if p_A appears after $\mathbf{exc}_0$, but not between $\mathbf{han}_0$ and $\mathbf{exc}_0$, unlike $\Diamond^d \mathrm{p}_A$. To avoid this, the path must be stopped earlier, by embedding an appropriate subformula as the left operand of the until.

Suppose there exists a formula ψ that is true in $\mathbf{exc}_0$, and false in all positions between $\mathbf{han}_0$ and $\mathbf{exc}_0$. By Lemma 3.2, there exists an integer n such that for any $w \in L_{\mathbf{call}}$ longer than n there is $w' = uv^k xy^k z \in L_{\mathbf{call}}$, for some $k > 0$, such that either (a) ψ never holds in $v^k xy^k$, or (b) it holds at least k times in there. We can take w such that $\mathbf{han}_0$ and $\mathbf{exc}_0$ both appear after position n, and they contain nested **han**–**exc** pairs. In case (a), ψ cannot distinguish $\mathbf{exc}_0$ from nested **exc**s, so a φ based on ψ is not equivalent to $\Diamond^d \mathrm{p}_A$ in $\mathbf{han}_0$. The same can be said in case (b), by evaluating φ in a **han** from v^i with $i < k$. In this case, also chaining multiple untils, each one ending in a position in which ψ holds, does not work, as k can be increased beyond the finite length of any OPTL formula.

The above argument holds verbatim for LTL Until, and does not change if we prepend LTL or abstract next operators to the until, because the length of the branch between $\mathbf{call}_1$ and $\mathbf{call}_0$ is unlimited. The argument for using since operators starting from $\mathbf{exc}_0$ is symmetric. If both until and since operators are used, it suffices to apply the Pumping Lemma twice (one for until and one for since), and take a value of k large enough that a part of the string cannot be reached by the number of until and since operators in the formula. If Hierarchical operators are used, the argument does not change, as they still need nested until or since operators to cover the whole subtree.

This argument also holds when the formula contains (possibly nested) negated until operators. This is trivial if their paths cannot reach part of the subtree between $\mathbf{han}_0$ and $\mathbf{exc}_0$. If, instead, they can reach positions past $\mathbf{exc}_0$, we can build a word with p_A in one of such positions, but not between $\mathbf{han}_0$ and $\mathbf{exc}_0$. To distinguish it from a word with p_A between $\mathbf{han}_0$ and $\mathbf{exc}_0$, the formula would need a subformula that can distinguish positions between $\mathbf{han}_0$ and $\mathbf{exc}_0$ from those outside, which would contradict Lemma 3.2.

Until now, we have proved that a formula equivalent to $\Diamond^d \mathrm{p}_A$ cannot contain until or since operators that stop exactly at $\mathbf{exc}_0$. However, they could be stopped earlier. In the following, we show that any such OPTL formula can only work for words of a limited length. Hence, no OPTL formula is equivalent to $\Diamond^d \mathrm{p}_A$ on all words in $L_{\mathbf{call}}$.

Let $w \in L_{\mathbf{call}}$, and $x = \mathbf{han}\, y\, \mathbf{exc}$ a subword of w with the structure of Figure 7, in which each **han** has a matched **exc**, and conversely. We define $\mathrm{h}_{\mathbf{exc}}(x) = 0$ if y contains no positions labeled with **han** or **exc** (hence, only **call**s and **ret**s). Otherwise, let $x' = \mathbf{han}\, y'\, \mathbf{exc}$ be the proper subword of y with the maximum value of $\mathrm{h}_{\mathbf{exc}}(x')$: we set $\mathrm{h}_{\mathbf{exc}}(x) = \mathrm{h}_{\mathbf{exc}}(x') + 1$.

We prove by induction on $\mathrm{h}_{\mathbf{exc}}(x)$ that any OPTL formula evaluated in the first position of x must contain nested until or since operators with a nesting depth of at least $2 \cdot \mathrm{h}_{\mathbf{exc}}(x)+1$ to be equivalent to $\Diamond^d \mathrm{p}_A$.

If $\mathrm{h}_{\mathbf{exc}}(x) = 0$, at least one until or since operator is needed, as the length of x is not fixed. E.g., OPTL formula $\neg \mathbf{exc}\, \mathcal{U}^{\lessdot \doteq \gtrdot}\, \mathrm{p}_A$ suffices.

If $\mathrm{h}_{\mathbf{exc}}(x) = n > 0$, Figure 7 shows a possible structure of x. Any Summary Until in the formula must be nested into another operator, or its paths would jump to, and go past, the last position of x ($\mathbf{exc}_0$). A Summary Until could be, instead, nested into any number of nested next operators, to be evaluated in one of the positions shown in Figure 7 between $\mathbf{han}_0$ and $\mathbf{call}_0$. (The tree fragments between $\mathbf{call}_1$ and $\mathbf{call}_0$ can be repeated enough times so that the next operators alone cannot reach $\mathbf{call}_0$.) As noted earlier, any such summary until must allow for paths with consecutive positions in the $\gtrdot$ relation. It may also jump to $\mathbf{exc}_0$ by following the chain relation, because $\mathbf{call} \gtrdot \mathbf{exc}$. Hence, the until must be stopped earlier by choosing appropriate operands (e.g., $\neg \bigcirc_\chi \mathbf{exc}$ as the left operand). However, this leaves the subword between $\mathbf{call}_0$ and $\mathbf{exc}_0$ unreached, so any of its positions containing (or not) p_A would be ignored. This can only be solved with another until operator, so at least two are needed. If it is a Summary Until, then it must not allow the $\gtrdot$ relation, or it could, again, escape $\mathbf{exc}_0$ (e.g. by skipping chains between $\mathbf{call}$s and $\mathbf{exc}_0$). The argument can be extended by considering an LTL Until which stops anywhere before $\mathbf{exc}_0$, or since operators evaluated in $\mathbf{exc}$ (e.g., nested in a $\bigcirc_\chi$ operator). The same can be said for Hierarchical operators, which can cover only a part of the subtree if used alone.

Let $x' = \mathbf{han}\, y'\, \mathbf{exc}$ be a proper subword of y with $\mathrm{h}_{\mathbf{exc}}(x') = n-1$. Suppose it appears before $\mathbf{call}_0$ (the other case is symmetric). It needs at least an until or since operator to be covered, which must not escape $\mathbf{han}_0$ or $\mathbf{exc}_0$. The Pumping Lemma can be used to show that no OPTL formula can distinguish positions in x or x' from those outside. Thus, a formula with until or since operators that do not exit y' is needed. By the inductive hypothesis, it consists of at least $2(n-1)+1$ until or since operators, thus x needs $2n+1$ of them.

Note that the argument also holds if the until formulas are negated, because negation cannot change the type of paths considered by an operator, and cannot decrease the number of nested untils needed to cover the whole subtree. □

4. First-Order Completeness on Finite Words

To show that POTL $\subseteq$ FOL on finite OP words, we give a direct translation of POTL into FOL. Proving that FOL $\subseteq$ POTL is more involved: we translate $\mathcal{X}_{until}$ [Mar04], a logic on trees, into POTL. $\mathcal{X}_{until}$ (defined in Section 4.2.2) is a logic on trees introduced to prove the expressive completeness of Conditional XPath, and from its being equivalent to FOL on trees [Mar05, LS10] we derive a FO-completeness result for POTL.

4.1. First-Order Semantics of POTL.

We show that POTL can be expressed with FOL equipped with monadic relations for atomic propositions, a total order on positions, and the chain relation between pairs of positions. We define below the translation function ν, such that for any POTL formula φ, word w and position i, $(w,i) \models \nu_\varphi(x)$ iff $(w,i) \models \varphi$. The translation for propositional operators is trivial.

For temporal operators, we first need to define a few auxiliary formulas. We define the successor relation as the FO formula

$$\mathrm{succ}(x, y) := x < y \wedge \neg\exists z(x < z \wedge z < y).$$

The PRs between positions can be expressed by means of propositional combinations of monadic atomic relations only. Given a set of atomic propositions $a \subseteq AP$, we define formula $\sigma_a(x)$, stating that all and only propositions in a hold in position x, as follows:

$$\sigma_a(x) := \bigwedge_{\mathrm{p} \in a} \mathrm{p}(x) \wedge \bigwedge_{\mathrm{p} \in AP \setminus a} \neg \mathrm{p}(x) \tag{4.1}$$

For any pair of FO variables x, y and $\pi \in \{\lessdot, \doteq, \gtrdot\}$, we can build formula

$$x \; \pi \; y := \bigvee_{a,b \subseteq AP \mid a \pi b} (\sigma_a(x) \wedge \sigma_b(y)).$$

The following translations employ the three FO variables x, y, z, only. This, in addition to the FO-completeness result for POTL, proves that FOL on OP words retains the three-variable property, which holds in regular words.

4.1.1. *Next and Back Operators.*

$$\nu_{\bigcirc^d \varphi}(x) := \exists y\Big(\mathrm{succ}(x, y) \wedge (x \lessdot y \vee x \doteq y) \wedge \exists x\big(x = y \wedge \nu_\varphi(x)\big)\Big)$$

$\nu_{\ominus^d \varphi}(x)$ is defined similarly, and $\nu_{\bigcirc^u \varphi}(x)$ and $\nu_{\ominus^u \varphi}(x)$ by replacing $\lessdot$ with $\gtrdot$.

$$\nu_{\chi_F^d \varphi}(x) := \exists y\big(\chi(x, y) \wedge (x \lessdot y \vee x \doteq y) \wedge \exists x(x = y \wedge \nu_\varphi(x))\big)$$

$\nu_{\chi_P^d \varphi}(x)$, $\nu_{\chi_F^u \varphi}(x)$ and $\nu_{\chi_P^u \varphi}(x)$ are defined similarly.

4.1.2. *Downward/Upward Summary Until/Since.* The translation for the DS until operator can be obtained by noting that, given two positions x and y, the DSP between them, if it exists, is the one that skips all chain bodies entirely contained between them, among those with contexts in the $\lessdot$ or $\doteq$ relations. A position z being part of such a path can be expressed with formula $\neg\gamma(x, y, z)$ as follows:

$$\begin{aligned}
\gamma(x, y, z) &:= \gamma_L(x, z) \wedge \gamma_R(y, z) \\
\gamma_L(x, z) &:= \exists y\Big(x \leq y \wedge y < z \wedge \exists x\big(z < x \wedge \chi(y, x) \wedge (y \lessdot x \vee y \doteq x)\big)\Big) \\
\gamma_R(y, z) &:= \exists x\Big(z < x \wedge x \leq y \wedge \exists y\big(y < z \wedge \chi(y, x) \wedge (y \lessdot x \vee y \doteq x)\big)\Big)
\end{aligned}$$

$\gamma(x, y, z)$ is true iff z is not part of the DSP between x and y, while $x \leq z \leq y$. In particular, $\gamma_L(x, z)$ asserts that z is part of the body of a chain whose left context is after x, and $\gamma_R(y, z)$ states that z is part of the body of a chain whose right context is before y. Since chain bodies cannot cross, either the two chain bodies are actually the same one, or one of them is a sub-chain nested into the other. In both cases, z is part of a chain body entirely contained between x and y, and is thus not part of the path.

Moreover, for such a path to exist, each one of its positions must be in one of the admitted PRs with the next one. Formula

$$\delta(y, z) := \exists x\big(z < x \wedge x \leq y \wedge (z \lessdot x \vee z \doteq x) \wedge \neg\gamma(z, y, x) \wedge (\mathrm{succ}(z, x) \vee \chi(z, x))\big)$$

asserts this for position z, with the path ending in y. (Note that by exchanging x and z in the definition of $\gamma(x,y,z)$ above, one can obtain $\gamma(z,y,x)$ without using any additional variable.) Finally, $\varphi\,\mathcal{U}^d_\chi\,\psi$ can be translated as follows:

$$\nu_{\varphi\mathcal{U}^d_\chi\psi}(x) := \exists y\Big(x \le y \wedge \exists x(x = y \wedge \nu_\psi(x)) \\ \wedge \forall z\big(x \le z \wedge z < y \wedge \neg\gamma(x,y,z) \implies \exists x(x = z \wedge \nu_\varphi(x)) \wedge \delta(y,z)\big)\Big)$$

The translation for the DS since operator is similar:

$$\nu_{\varphi\mathcal{S}^d_\chi\psi}(x) := \exists y\Big(y \le x \wedge \exists x(x = y \wedge \nu_\psi(x)) \\ \wedge \forall z\big(y < z \wedge z \le x \wedge \neg\gamma(y,x,z) \implies \exists x(x = z \wedge \nu_\varphi(x)) \wedge \delta(x,z)\big)\Big)$$

$\nu_{\varphi\mathcal{U}^u_\chi\psi}(x)$ and $\nu_{\varphi\mathcal{S}^u_\chi\psi}(x)$ are defined as above, substituting $\succ$ for $\prec$.

4.1.3. *Hierarchical Operators.* Finally, below are the translations for two hierarchical operators, the others being symmetric.

$$\nu_{\bigcirc^u_H\varphi}(x) := \exists y\Bigg(y < x \wedge \chi(y,x) \wedge y \lessdot x \wedge \\ \exists z\Big(x < z \wedge \chi(y,z) \wedge y \lessdot z \wedge \exists x(x = z \wedge \nu_\varphi(x)) \\ \wedge \forall y\big(x < y \wedge y < z \implies \forall z(\chi(z,x) \wedge z \lessdot x \implies \neg\chi(z,y))\big)\Big)\Bigg)$$

$$\nu_{\varphi\mathcal{U}^u_H\psi}(x) := \exists z\Bigg(z < x \wedge z \lessdot x \wedge \chi(z,x) \wedge \\ \exists y\Big(x \le y \wedge \chi(z,y) \wedge z \lessdot y \wedge \exists x(x = y \wedge \nu_\psi(x)) \wedge \\ \forall z\big(x \le z \wedge z < y \wedge \exists y(y < x \wedge y \lessdot x \wedge \chi(y,x) \wedge \chi(y,z)) \\ \implies \exists x(x = z \wedge \nu_\varphi(x))\big)\Big)\Bigg)$$

4.2. **Expressing $\mathcal{X}_{until}$ in POTL.** To translate $\mathcal{X}_{until}$ to POTL, we give an isomorphism between OP words and (a subset of) unranked ordered trees (UOT), the structures on which $\mathcal{X}_{until}$ is defined. First, we show how to translate OP words into UOTs, and then the reverse.

4.2.1. *OPM-compatible Unranked Ordered Trees.*

Definition 4.1 (Unranked Ordered Trees)**.** A UOT is a tuple $T = \langle S, R_\Downarrow, R_\Rightarrow, L\rangle$. Each node is a sequence of child numbers, representing the path from the root to it. S is a finite set of finite sequences of natural numbers closed under the prefix operation, and for any sequence $s \in S$, if $s \cdot k \in S$, $k \in \mathbb{N}$, then either $k = 0$ or $s \cdot (k-1) \in S$ (by $\cdot$ we denote concatenation). $R_\Downarrow$ and $R_\Rightarrow$ are two binary relations called the *descendant* and *following sibling* relation, respectively. For $s, t \in S$, $sR_\Downarrow t$ iff t is any child of s ($t = s \cdot k$, $k \in \mathbb{N}$, i.e. t is the k-th child of s), and $sR_\Rightarrow t$ iff t is the immediate sibling to the right of s ($s = r \cdot h$ and

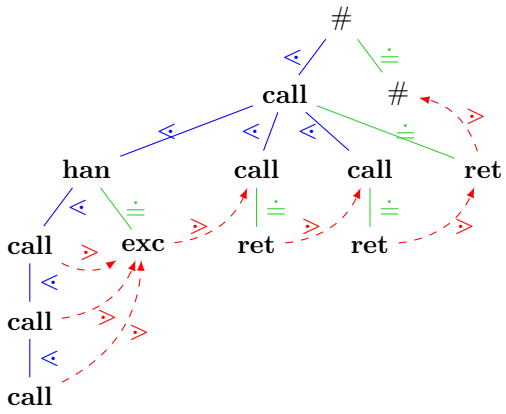

Figure 8: The UOT corresponding to the word of Figure 5, and the ST of Figure 3. PRs between adjacent nodes are highlighted: $\lessdot$ is blue and $\doteq$ is green. A dashed red arrow connects each node to its Rc, if any.

$t = r \cdot (h+1)$, for $r \in S$ and $h \in \mathbb{N}$). $L\colon AP \to \mathcal{P}(S)$ is a function that maps each atomic proposition to the set of nodes labeled with it. We denote as $\mathcal{T}$ the set of all UOTs.

Given an OP word $w = \langle U, <, M_{\mathcal{P}(AP)}, P\rangle$, it is possible to build a UOT $T_w = \langle S_w, R_{\Downarrow}, R_{\Rightarrow}, L_w\rangle \in \mathcal{T}$ with labels in $\mathcal{P}(AP)$ isomorphic to w. To do so, we define a function $\tau\colon U \to S_w$, which maps positions of w into nodes of T_w. First, $\tau(0) = 0$: position 0 is the root node, and the last # is its rightmost child. Given any position $i \in U$:

- if $i \doteq i+1$, then $\tau(i+1) = \tau(i)\cdot 0$ is the only child of i;
- if $i \gtrdot i+1$, then i has no children;
- if $i \lessdot i+1$, then the leftmost child of i is $i+1$ ($\tau(i+1) = \tau(i)\cdot 0$). Moreover, if $j_1 < j_2 < \cdots < j_n$ is the largest set of positions such that $\chi(i, j_k)$ and either $i \lessdot j_k$ or $i \doteq j_k$ for $1 \le k \le n$, then $\tau(j_k) = \tau(i)\cdot k$.

In general, i is in the $\lessdot$ relation with all of its children, except possibly the rightmost one, with which i may be in the $\doteq$ relation (cf. property 4 of the χ relation).

This way, every position i in w appears in the UOT exactly once. Indeed, if either $(i-1) \lessdot i$ or $(i-1) \doteq i$, then i is a child of $i-1$. Conversely, $(i-1) \gtrdot i$ iff i is the right context of at least one chain. Thus, consider j s.t. $\chi(j, i)$, and for no $j' < j$ we have $\chi(j', i)$: by property 3 of χ, either $j \doteq i$ or $j \lessdot i$. So, i is a child of j by the third rule above, and of no other node, because if $(i-1) \gtrdot i$, then no other rule applies.

Finally, $\tau(i) \in L_w(\mathrm{a})$ iff $i \in P(\mathrm{a})$ for all $\mathrm{a} \in AP$, so each node in T_w is labeled with the set of atomic propositions that hold in the corresponding word position. We denote as $T_w = \tau(w)$ the UOT obtained by applying τ to every position of an OP word w. Figure 8 shows the translation of the word of Figure 5 into an UOT.

As for the other way of the isomorphism, notice that we are considering only a subset of UOTs. In fact, we only consider UOTs whose node labels are compatible with a given OPM $M_{\mathcal{P}(AP)}$. In order to define the notion of OPM compatibility for UOTs, we need to introduce the *right context* (Rc) of a node. Given a UOT T and a node $s \in T$, the Rc of s is denoted $\mathrm{Rc}(s)$. If s has a child s' such that $s \doteq s'$, then $\mathrm{Rc}(s)$ is undefined. Otherwise, if r is the leftmost right sibling of s, then $\mathrm{Rc}(s) = r$; if s has no right siblings, then $\mathrm{Rc}(s) = \mathrm{Rc}(p)$, where p is the parent of s. In Figure 8, nodes are linked to their Rc by a dashed red arrow.

In the following, for any nodes s, s' and $\pi \in \{\lessdot, \doteq, \gtrdot\}$, we write $s\ \pi\ s'$ meaning that $a\ \pi\ b$, where $a = \{\mathrm{p} \mid s \in L(\mathrm{p})\}$, and $b = \{\mathrm{p} \mid s' \in L(\mathrm{p})\}$.

Definition 4.2 (OPM-compatible UOTs)**.** We denote the set of UOTs compatible with an OPM M as $\mathcal{T}_M$. A UOT T is in $\mathcal{T}_M$ iff the following properties hold. The root node and its rightmost child are the only ones labeled with #. For any node $s \in T$, its rightmost child r, if any, is such that either $s \lessdot r$ or $s \doteq r$. For any other child $s' \neq r$ of s, we have $s \lessdot s'$. If $\mathrm{Rc}(s)$ exists, then $s \gtrdot \mathrm{Rc}(s)$.

Given a tree $T \in \mathcal{T}_M$ with labels on $\mathcal{P}(AP)$, it is possible to build an OP word w_T isomorphic to T. Indeed,

Lemma 4.3. *Given an OP word w and the UOT $T_w = \tau(w)$, function τ is an isomorphism between positions of w and nodes of T_w.*

Proof. We define function $\tau_{AP}^{-1} : S \to \mathcal{P}(AP)^+$, which maps a UOT node to the subword corresponding to the subtree rooted in it. For any node $s \in T$, let its label $a = \{\mathrm{p} \mid s \in L(\mathrm{p})\}$, and let $c_0, c_1 \dots c_n$ be its children, if any. $\tau_{AP}^{-1}(s)$ is defined as $\tau_{AP}^{-1}(s) = a$ if s has no children, and $\tau_{AP}^{-1}(s) = a \cdot \tau_{AP}^{-1}(c_0) \cdot \tau_{AP}^{-1}(c_1) \cdots \tau_{AP}^{-1}(c_n)$ otherwise. We prove $\tau_{AP}^{-1}(s)$ is an OP word.

We need to prove by induction on the tree structure that for any tree node s, $\tau_{AP}^{-1}(s)$ is of the form $a_0 x_0 a_1 x_1 \dots a_n x_n$, with $n \geq 0$, and such that for $0 \leq k < n$, $a_k \doteq a_{k+1}$ and either $x_k = \varepsilon$ or ${}^{a_k}[x_k]^{a_{k+1}}$. In the following, we denote as $\mathrm{first}(x)$ the first position of a string x, and as $\mathrm{last}(x)$ the last one. Indeed, for each $0 \leq i < n$ we have $a \lessdot \mathrm{first}(\tau_{AP}^{-1}(c_i))$, and the rightmost leaf f_i of the tree rooted in c_i is such that $\mathrm{Rc}(f_i) = c_{i+1}$. Since $f_i = \tau(\mathrm{last}(\tau_{AP}^{-1}(c_i)))$ and $c_{i+1} = \tau(\mathrm{first}(\tau_{AP}^{-1}(c_{i+1})))$, we have $\mathrm{last}(\tau_{AP}^{-1}(c_i)) \gtrdot \mathrm{first}(\tau_{AP}^{-1}(c_{i+1}))$. So, ${}^{a}[\tau_{AP}^{-1}(c_i)]^{\mathrm{first}(\tau_{AP}^{-1}(c_{i+1}))}$. As for $\tau_{AP}^{-1}(c_n)$, if $a \lessdot c_n$ then $\tau_{AP}^{-1}(s) = a_0 x_0$ (and $a_0 \lessdot \mathrm{first}(x_0)$), with $a_0 = a$ and $x_0 = \tau_{AP}^{-1}(c_0) \cdot \tau_{AP}^{-1}(c_1) \cdots \tau_{AP}^{-1}(c_n)$. If $a \doteq c_n$, consider that, by hypothesis, $\tau_{AP}^{-1}(c_n)$ is of the form $a_1 x_1 a_2 \dots a_n x_n$. So $\tau_{AP}^{-1}(s) = a_0 x_0 a_1 x_1 a_2 \dots a_n x_n$, with $a_0 = a$ and $x_0 = \tau_{AP}^{-1}(c_0) \cdot \tau_{AP}^{-1}(c_1) \cdots \tau_{AP}^{-1}(c_{n-1})$.

The root 0 of T and its rightmost child $c_\#$ are labeled with #. So, $\tau_{AP}^{-1}(c_\#) = \#$, and $\tau_{AP}^{-1}(0) = \#x_0\#$, with ${}^{\#}[x_0]^{\#}$, which is a finite OP word.

$\tau^{-1} : S \to U$ can be derived from τ_{AP}^{-1}. By construction, we have $\tau^{-1}(\tau(i)) = i$ for any word w and position i. □

From Lemma 4.3 follows:

Proposition 4.4. *Let M_{AP} be an OPM on $\mathcal{P}(AP)$. For any FO formula $\varphi(x)$ on OP words compatible with M_{AP}, there exists a FO formula $\varphi'(x)$ on trees in $\mathcal{T}_{M_{AP}}$ such that for any OP word w and position i in it, $w \models \varphi(i)$ iff $T_w \models \varphi'(\tau(i))$, with $T_w = \tau(w)$.*

4.2.2. *POTL Translation of $\mathcal{X}_{until}$.* We now give the full translation of the logic $\mathcal{X}_{until}$ from [Mar04] into POTL.

The syntax of $\mathcal{X}_{until}$ formulas is $\varphi ::= \mathrm{a} \mid \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \rho(\varphi, \varphi)$, with $\mathrm{a} \in AP$ and $\rho \in \{\Downarrow, \Uparrow, \Rightarrow, \Leftarrow\}$. The semantics of propositional operators is the usual one, while $\rho(\varphi, \varphi)$ is an until/since operator on the child and sibling relations. Let $T \in \mathcal{T}$ be a UOT with nodes in S. For any $r, s \in S$, $R_\Uparrow$ and $R_\Leftarrow$ are s.t. $rR_\Uparrow s$ iff $sR_\Downarrow r$, and $rR_\Leftarrow s$ iff $sR_\Rightarrow r$. We denote as R_ρ^+ the transitive (but not reflexive) closure of relation R_ρ, and by R_ρ^* its transitive *and* reflexive closure. For $s \in S$, $(T, s) \models \rho(\varphi, \psi)$ iff there exists a node $t \in S$ s.t. $sR_\rho^+ t$ and

$(T,t) \models \psi$, and for any $r \in S$ s.t. $sR^+_\rho r$ and $rR^+_\rho t$ we have $(T,r) \models \varphi$. Notice that $t \neq s$ and $r \neq s$, so s is not included in the paths: we call this semantics *strict*. Conversely, in POTL paths always start from the position where an until/since operator is evaluated.

[LS10] proved the equivalence of $\mathcal{X}_{until}$ to the logic Conditional XPath, which was proved equivalent to FOL on finite UOTs in [Mar05]. This result is valid for any labeling of tree nodes, and so is on OPM-compatible UOTs.

Theorem 4.5 ([Mar05, LS10])**.** *Let M_{AP} be an OPM on AP. For any FO formula $\varphi(x)$ on trees in $\mathcal{T}_{M_{AP}}$, there exists a $\mathcal{X}_{until}$ formula φ' such that, for any $T \in \mathcal{T}_{M_{AP}}$ and node $t \in T$, we have $T \models \varphi(t)$ iff $(T,t) \models \varphi'$.*

We define function $\iota_{\mathcal{X}}$, which translates any $\mathcal{X}_{until}$ formula φ into a POTL formula s.t. φ holds on a UOT T iff $\iota_{\mathcal{X}}(\varphi)$ holds on the isomorphic word w_T. $\iota_{\mathcal{X}}$ is defined as the identity for the propositional operators, and with the equivalences below for the other $\mathcal{X}_{until}$ operators. Recall from Section 3.4.1 that for any $a \subseteq AP$, $\sigma_a := \bigwedge_{\mathrm{p}\in a} \mathrm{p} \wedge \bigwedge_{\mathrm{q}\notin a} \neg \mathrm{q}$ holds in a position i iff a is the set of atomic propositions holding in i. For any POTL formula γ, let $\chi^{\lessdot}_F\gamma := \bigvee_{a,b\subseteq AP,\, a<b}(\sigma_a \wedge \chi^d_F(\sigma_b \wedge \gamma))$ be the restriction of $\chi^d_F\gamma$ to chains with contexts in the $\lessdot$ PR; operators $\chi^{\doteq}_F\gamma$, $\chi^{\lessdot}_P\gamma$, $\chi^{\doteq}_P\gamma$, $\bigcirc^{\lessdot}\gamma$, $\ominus^{\lessdot}\gamma$ are defined analogously.

For any $\mathcal{X}_{until}$ formulas φ, ψ, let $\varphi' = \iota_{\mathcal{X}}(\varphi)$ and $\psi' = \iota_{\mathcal{X}}(\psi)$. We define $\iota_{\mathcal{X}}$ as follows:

$$\iota_{\mathcal{X}}(\Downarrow(\varphi,\psi)) := \bigcirc^d(\varphi' \,\mathcal{U}^d_\chi\, \psi') \vee \chi^d_F(\varphi' \,\mathcal{U}^d_\chi\, \psi') \tag{4.2}$$

$$\iota_{\mathcal{X}}(\Uparrow(\varphi,\psi)) := \ominus^d(\varphi' \,\mathcal{S}^d_\chi\, \psi') \vee \chi^d_P(\varphi' \,\mathcal{S}^d_\chi\, \psi') \tag{4.3}$$

$$\iota_{\mathcal{X}}(\Rightarrow(\varphi,\psi)) := \bigcirc^u_H(\varphi' \,\mathcal{U}^u_H\, \psi') \tag{4.4}$$

$$\vee \big(\neg \bigcirc^u_H(\top \,\mathcal{U}^u_H\, \neg\varphi') \wedge \chi^{\lessdot}_P(\chi^{\doteq}_F\psi')\big) \tag{4.5}$$

$$\vee \ominus^{\lessdot}\Big(\chi^{\lessdot}_F\big(\psi' \wedge \neg \ominus^u_H(\top \,\mathcal{S}^u_H\, \neg\varphi')\big)\Big) \tag{4.6}$$

$$\vee \ominus^{\lessdot}(\chi^{\doteq}_F\psi' \wedge \neg\chi^{\lessdot}_F\neg\varphi') \tag{4.7}$$

$$\iota_{\mathcal{X}}(\Leftarrow(\varphi,\psi)) := \ominus^u_H(\varphi' \,\mathcal{S}^u_H\, \psi') \tag{4.8}$$

$$\vee \chi^{\doteq}_P\big(\chi^{\lessdot}_F(\neg \bigcirc^u_H \top \wedge \varphi' \,\mathcal{S}^u_H\, \psi')\big) \tag{4.9}$$

$$\vee \big(\chi^{\lessdot}_P(\bigcirc^{\lessdot}\psi') \wedge \neg \ominus^u_H(\top \,\mathcal{S}^u_H\, \neg\varphi')\big) \tag{4.10}$$

$$\vee \chi^{\doteq}_P(\bigcirc^{\lessdot}\psi' \wedge \neg\chi^{\lessdot}_F\neg\varphi') \tag{4.11}$$

We prove the correctness of this translation in the following lemmas.

Lemma 4.6. *Given an OP alphabet (AP, M_{AP}), for every $\mathcal{X}_{until}$ formula $\Downarrow(\varphi,\psi)$, and for any OP word w and position i in w, we have*

$$(T_w, \tau(i)) \models \Downarrow(\varphi,\psi) \text{ iff } (w,i) \models \iota_{\mathcal{X}}(\Downarrow(\varphi,\psi)).$$

$T_w \in \mathcal{T}_{M_{AP}}$ is the UOT obtained by applying function τ to every position in w, such that for any position i' in w $(T_w, \tau(i')) \models \varphi$ iff $(w,i') \models \iota_{\mathcal{X}}(\varphi)$, and likewise for ψ.

Proof. Let $\varphi' = \iota_{\mathcal{X}}(\varphi)$ and $\psi' = \iota_{\mathcal{X}}(\psi)$.

[Only if] Suppose $(T_w, \tau(i)) \models \Downarrow(\varphi,\psi)$. Let $r = \tau(i)$, and $s = \tau(j)$ s.t. $rR_{\Downarrow}s$ and s is the first tree node of the path witnessing $\Downarrow(\varphi,\psi)$.

We inductively prove that $\varphi'\mathcal{U}^d_\chi\psi'$ holds in j. If s is the last node of the path, then ψ' holds in j and so does, trivially, $\varphi'\mathcal{U}^d_\chi\psi'$. Otherwise, consider any node $t = \tau(k)$ in the path, except

the last one, and suppose $\varphi' \mathcal{U}^d_\chi \psi'$ holds in k' s.t. $t' = \tau(k')$ is the next node in the path. If t' is the leftmost child of t, then $k' = k+1$ and either $k \lessdot k'$ or $k \doteq k'$: in both cases $\bigcirc^d(\varphi' \mathcal{U}^d_\chi \psi')$ holds in k. If t' is not the leftmost child, then $\chi(k, k')$ and $k \lessdot k'$ or $k \doteq k'$: so $\chi^d_F(\varphi' \mathcal{U}^d_\chi \psi')$ holds in k. Thus, by expansion law $\varphi' \mathcal{U}^d_\chi \psi' \equiv \psi' \vee (\varphi' \wedge (\bigcirc^d(\varphi' \mathcal{U}^d_\chi \psi') \vee \chi^d_F(\varphi' \mathcal{U}^d_\chi \psi')))$, $\varphi' \mathcal{U}^d_\chi \psi'$ holds in k and, by induction, also in j.

Suppose s is the leftmost child of r: $j = i+1$, and either $i \lessdot j$ or $i \doteq j$, so $\bigcirc^d(\varphi' \mathcal{U}^d_\chi \psi')$ holds in i. Otherwise, $\chi(i, j)$ and either $i \lessdot j$ or $i \doteq j$. In both cases, $\chi^d_F(\varphi' \mathcal{U}^d_\chi \psi')$ holds in i.

[If] Suppose (4.2) holds in i. If $\bigcirc^d(\varphi' \mathcal{U}^d_\chi \psi')$ holds in i, then $\varphi' \mathcal{U}^d_\chi \psi'$ holds in $j = i+1$, and either $i \lessdot j$ or $i \doteq j$: then $s = \tau(j)$ is the leftmost child of $\tau(i)$. If $\chi^d_F(\varphi' \mathcal{U}^d_\chi \psi')$ holds in i, then $\varphi' \mathcal{U}^d_\chi \psi'$ holds in j s.t. $\chi(i, j)$ and $i \lessdot j$ or $i \doteq j$: $s = \tau(j)$ is a child of $\tau(i)$ in this case as well.

We prove that if $\varphi' \mathcal{U}^d_\chi \psi'$ holds in a position j s.t. $\tau(i) R_\Downarrow \tau(j)$, then $\Downarrow(\varphi, \psi)$ holds in $\tau(i)$. If $\varphi' \mathcal{U}^d_\chi \psi'$ holds in j, then there exists a DSP of minimal length from j to $h > j$ s.t. $(w, h) \models \psi'$ and φ' holds in all positions $j \leq k < h$ of the path, and $(T_w, \tau(k)) \models \varphi$. In any such k, $\varphi' \mathcal{U}^d_\chi \psi' \equiv \psi' \vee (\varphi' \wedge (\bigcirc^d(\varphi' \mathcal{U}^d_\chi \psi') \vee \chi^d_F(\varphi' \mathcal{U}^d_\chi \psi')))$ holds. Since this DSP is the minimal one, ψ' does not hold in k. Either $\bigcirc^d(\varphi' \mathcal{U}^d_\chi \psi')$ or $\chi^d_F(\varphi' \mathcal{U}^d_\chi \psi')$ hold in it. Therefore, the next position in the path is k' s.t. either $k' = k+1$ or $\chi(k, k')$, and either $k \lessdot k'$ or $k \doteq k'$, and $(w, k') \models \varphi' \mathcal{U}^d_\chi \psi'$. Therefore, $\tau(k')$ is a child of $\tau(k)$. So, there is a sequence of nodes $s_0, s_1, \ldots, s_n$ in T_w s.t. $\tau(i) R_\Downarrow s_0$, and $s_i R_\Downarrow s_{i+1}$ and $(T_w, s_i) \models \varphi$ for $0 \leq i < n$, and $(T_w, s_n) \models \psi$. This is a path making $\Downarrow(\varphi, \psi)$ true in $\tau(i)$. □

The proof for $\iota_\mathcal{X}(\Uparrow(\varphi, \psi))$ (4.3) is analogous to Lemma 4.6, and is therefore omitted.

Lemma 4.7. *Given an OP alphabet (AP, M_{AP}), for every $\mathcal{X}_{until}$ formula $\Rightarrow(\varphi, \psi)$, and for any OP word w and position i in w, we have*

$$(T_w, \tau(i)) \models \Rightarrow(\varphi, \psi) \text{ iff } (w, i) \models \iota_\mathcal{X}(\Rightarrow(\varphi, \psi)).$$

$T_w \in \mathcal{T}_{M_{AP}}$ is the UOT obtained by applying function τ to every position in w, such that for any position i' in w $(T_w, \tau(i')) \models \varphi$ iff $(w, i') \models \iota_\mathcal{X}(\varphi)$, and likewise for ψ.

Proof. Let $\varphi' = \iota_\mathcal{X}(\varphi)$ and $\psi' = \iota_\mathcal{X}(\psi)$.

[Only if] Suppose $\Rightarrow(\varphi, \psi)$ holds in $s = \tau(i)$. Then, node $r = \tau(h)$ s.t. $r R_\Downarrow s$ has at least two children, and $\Rightarrow(\varphi, \psi)$ is witnessed by a path starting in $t = \tau(j)$ s.t. $s R_\Rightarrow t$, and ending in $v = \tau(k)$. We have the following cases:

(1) *s is not the leftmost child of r.*
   (a) $h \lessdot k$. By the construction of T_w, for any node t' in the path, there exists a position $j' \in w$ s.t. $t' = \tau(j')$, $\chi(h, j')$ and $h \lessdot j'$. The path made by such positions is a UHP, and $\varphi' \mathcal{U}^u_H \psi'$ is true in j. Since s is not the leftmost child of r, we have $\chi(h, i)$, and $h \lessdot i$, so (4.4) holds in i.
   (b) *$h \doteq k$, so v is the rightmost child of r.* φ holds in all siblings between s and v (excluded), and φ' holds in the corresponding positions of w. All such positions j, if any, are s.t. $\chi(h, j)$ and $h \lessdot j$, and they form a UHP, so $\bigcirc^u_H(\top \mathcal{U}^u_H \neg\varphi')$ never holds in i. Moreover, since ψ holds in v, ψ' holds in k. Note that $\chi^{\lessdot}_P$ in i uniquely identifies position h, and $\chi^{\doteq}_F$ evaluated in h identifies k. So, (4.5) holds in i.

(2) *s is the leftmost child of r.* In this case, we have $i = h+1$ and $h \lessdot i$ (if $h \doteq i$, then r would have only one child).

(a) $h \lessdot k$. $\ominus^{\lessdot}$ evaluated in i identifies position h. ψ' holds in k, and $\ominus^u_H(\top\, \mathcal{S}^u_H\, \neg\varphi')$ does not, because in all positions between i and k (excluded) corresponding to children of r, φ' holds. Note that all such positions form a UHP, but i is not part of it ($i = h+1$, so $\neg\chi(h,i)$), and is not considered by $\top\, \mathcal{S}^u_H\, \neg\varphi'$. So, (4.6) holds in i.

(b) *$h \doteq k$, so v is the rightmost child of r.* ψ holds in v, and φ holds in all children of r, except possibly the first (s) and the last one (v). These are exactly all positions s.t. $\chi(h,j)$ and $h \lessdot j$. Since φ' holds in all of them by hypothesis, $\neg\chi^{\lessdot}_F\neg\varphi'$ holds in h. Since ψ holds in v, ψ' holds in k, and $\chi^{\doteq}_F\psi'$ in h. So, (4.7) holds in i.

[If] We separately consider cases (4.4)–(4.7).

(4.4): $\bigcirc^u_H(\varphi'\, \mathcal{U}^u_H\, \psi')$ holds in a position i in w. Then, there exists a position h s.t. $\chi(h,i)$ and $h \lessdot i$, and a position j s.t. $\chi(h,j)$ and $h \lessdot j$ that is the hierarchical successor of i, and $\varphi'\, \mathcal{U}^u_H\, \psi'$ holds in j. So, i and j are consecutive children of $r = \tau(h)$. Moreover, there exists a UHP between j and a position $k \geq j$. The tree nodes corresponding to all positions in the path are consecutive children of r, so we fall in case 1a of the *only if* part of the proof. In T_w, a path between $t = \tau(j)$ and $v = \tau(k)$ witnesses the truth of $\Rightarrow(\varphi,\psi)$ in s.

(4.5): $\neg \bigcirc^u_H (\top\, \mathcal{U}^u_H\, \neg\varphi' \wedge \chi^{\lessdot}_P(\chi^{\doteq}_F\psi'))$ holds in position $i \in w$ (this corresponds to case 1b). If $\chi^{\lessdot}_P(\chi^{\doteq}_F\psi')$ holds in i, then there exists a position h s.t. $\chi(h,i)$ and $h \lessdot i$, and a position k s.t. $\chi(h,k)$ and $h \doteq k$, and ψ' holds in k. $v = \tau(k)$ is the rightmost child of $r = \tau(h)$, parent of $s = \tau(i)$. Moreover, if $\neg \bigcirc^u_H (\top\, \mathcal{U}^u_H\, \neg\varphi')$ holds in i, then either:

- $\neg \bigcirc^u_H \top$ holds, i.e. there is no position $j > i$ s.t. $\chi(h,j)$ and $h \lessdot j$, so v is the immediate right sibling of s. In this case $\Rightarrow(\varphi,\psi)$ holds in s because ψ holds in v.
- $\neg(\top\, \mathcal{U}^u_H\, \neg\varphi')$ holds in $j > i$, the first position after i s.t. $\chi(h,j)$ and $h \lessdot j$. This means φ' holds in all positions $j' \geq j$ s.t. $\chi(h,j')$ and $h \lessdot j'$. Consequently, the tree nodes corresponding to these positions plus $v = \tau(k)$ form a path witnessing $\Rightarrow(\varphi,\psi)$, which holds in $s = \tau(i)$.

(4.6): $\ominus^{\lessdot}\Big(\chi^{\lessdot}_F\big(\psi' \wedge \neg \ominus^u_H (\top\, \mathcal{S}^u_H\, \neg\varphi')\big)\Big)$ holds in i. Let $h = i-1$, with $h \lessdot i$ (it exists because $\ominus^{\lessdot}$ is true). There exists a position k, $\chi(h,k)$ and $h \lessdot k$, in which ψ' holds, so ψ does in $v = \tau(k)$, and $\ominus^u_H(\top\, \mathcal{S}^u_H\, \neg\varphi')$ is false in it. If it is false because $\neg \ominus^u_H \top$ holds, there is no position $j < k$ s.t. $\chi(h,j)$ and $h \lessdot j$, so v is the second child of $r = \tau(h)$, $s = \tau(i)$ being the first one. So, $\Rightarrow(\varphi,\psi)$ trivially holds in s because ψ holds in the next sibling. Otherwise, let $j < k$ be the rightmost position lower than k s.t. $\chi(h,j)$ and $h \lessdot j$. $\neg(\top\, \mathcal{S}^u_H\, \neg\varphi')$ holds in it, so φ' holds in all positions j' between i and k that are part of the hierarchical path, i.e. s.t. $\chi(h,j')$ and $h \lessdot j'$. The corresponding tree nodes form a path ending in $v = \tau(k)$ that witnesses the truth of $\Rightarrow(\varphi,\psi)$ in s (case 2a).

(4.7): $\ominus^{\lessdot}(\chi^{\doteq}_F\psi' \wedge \neg\chi^{\lessdot}_F\neg\varphi')$ holds in i. Then let $h = i-1$, $h \lessdot i$, and $s = \tau(i)$ is the leftmost child of $r = \tau(h)$. Since $\chi^{\doteq}_F\psi'$ holds in h, there exists a position k, s.t. $\chi(h,k)$ and $h \doteq k$, in which ψ' holds. So, ψ holds in $v = \tau(k)$, which is the rightmost child of r, by construction. Moreover, φ' holds in all positions s.t. $\chi(h,j)$ and $h \lessdot j$. Hence, φ holds in all corresponding nodes $t = \tau(j)$, which are all nodes between s and v, excluded. This, together with ψ holding in v, makes a path that verifies $\Rightarrow(\varphi,\psi)$ in s (case 2b). □

Lemma 4.8. *Given an OP alphabet (AP, M_{AP}), for every $\mathcal{X}_{until}$ formula $\Leftarrow(\varphi,\psi)$, and for any OP word w and position i in w, we have*

$$(T_w, \tau(i)) \models \Leftarrow(\varphi,\psi) \text{ iff } (w,i) \models \iota_{\mathcal{X}}(\Leftarrow(\varphi,\psi)).$$

$T_w \in \mathcal{T}_{M_{AP}}$ is the UOT obtained by applying function τ to every position in w, such that for any position i' in w $(T_w, \tau(i')) \models \varphi$ iff $(w, i') \models \iota_{\mathcal{X}}(\varphi)$, and likewise for ψ.

Proof. Let $\varphi' = \iota_{\mathcal{X}}(\varphi)$ and $\psi' = \iota_{\mathcal{X}}(\psi)$.

[Only if] Suppose $\Leftarrow(\varphi, \psi)$ holds in $s = \tau(i)$. Then node $r = \tau(h)$ s.t. $rR_{\Downarrow}s$ has at least two children, and $\Leftarrow(\varphi, \psi)$ is true because of a path starting in $v = \tau(k)$, s.t. $rR_{\Downarrow}v$ and $(T_w, v) \models \psi$ and ending in $t = \tau(j)$ s.t. $tR_{\Rightarrow}s$. We distinguish between the following cases:

(1) *v is not the leftmost child of r.*
   (a) $h \lessdot i$. By construction, all nodes in the path correspond to positions $j' \in w$ s.t. $\chi(h, j')$ and $h \lessdot j'$, so they form a UHP. Hence, $\varphi' \, \mathcal{S}^u_H \, \psi'$ holds in j, and (4.8) holds in i.
   (b) $h \doteq i$. In this case, s is the rightmost child of r, and $\chi(h, i)$. The path made of positions between k and j corresponding to nodes between v and t (included) is a UHP. So $\varphi' \, \mathcal{S}^u_H \, \psi'$ holds in j, which is the rightmost position of any possible such UHP: so $\neg \bigcirc^u_H \top$ also holds in j. Hence, (4.9) holds in i.

(2) *v is the leftmost child of r.*
   (a) $h \lessdot i$. In this case, $k = h + 1$ and ψ' holds in k. So, $\bigcirc^{\lessdot}\psi'$ holds in h, and $\chi^{\leqslant}_P(\bigcirc^{\lessdot}\psi')$ holds in i. Moreover, in all word positions j' with $k < j' < j$ corresponding to children of r, φ' holds. Such positions form a UHP. So $\neg \ominus^u_H (\top \, \mathcal{S}^u_H \, \neg\varphi')$ holds in i. Note that this is true even if s is the first right sibling of v. Thus, (4.10) holds in i.
   (b) $h \doteq i$. ψ' holds in $k = h + 1$, so $\bigcirc^{\lessdot}\psi'$ holds in h. Since $\chi(h, i)$ and $h \doteq i$, $\chi^{\doteq}_P(\bigcirc^{\lessdot}\psi')$ holds in i. Moreover, φ holds in all children of r except the first and last one, so φ' holds in all positions j' s.t. $\chi(h, j')$ and $h \lessdot j'$. So $\neg\chi^{\leqslant}_F\neg\varphi'$ holds in h, and (4.11) in i.

[If] We separately consider cases (4.8)–(4.11).

(4.8): $\ominus^u_H(\varphi' \, \mathcal{S}^u_H \, \psi')$ holds in i. Then, there exists a position h s.t. $\chi(h, i)$ and $h \lessdot i$, and a position $j < i$ s.t. $\chi(h, j)$ and $h \lessdot j$. Since $j \neq h + 1$, the corresponding tree node is not the leftmost one. So, this corresponds to case 1a, and $\Leftarrow(\varphi, \psi)$ holds in $s = \tau(i)$.

(4.9): $\chi^{\doteq}_P\big(\chi^{\leqslant}_F(\neg \bigcirc^u_H \top \wedge \varphi' \, \mathcal{S}^u_H \, \psi')\big)$ holds in i. Then, there exists a position h s.t. $\chi(h, i)$ and $h \doteq i$. Moreover, at least a position j' s.t. $\chi(h, j')$ and $h \lessdot j'$ exists. Let j be the rightmost one, i.e. the only one in which $\neg \bigcirc^u_H \top$ holds. The corresponding tree node $t = \tau(j)$ is s.t. $tR_{\Rightarrow}s$, with $s = \tau(i)$. Since $\varphi' \, \mathcal{S}^u_H \, \psi'$ holds in j, a UHP starts from it, and ψ and φ hold in the tree nodes corresponding to, respectively, the first and all other positions in the path. This is case 1b, and $\Leftarrow(\varphi, \psi)$ holds in s.

(4.10): $\chi^{\leqslant}_P(\bigcirc^{\lessdot}\psi') \wedge \neg \ominus^u_H (\top \, \mathcal{S}^u_H \, \neg\varphi')$ holds in i. Then, there exists a position h s.t. $\chi(h, i)$ and $h \lessdot i$. ψ' holds in $k = h + 1$, so ψ holds in the leftmost child of $r = \tau(h)$. Moreover, φ' holds in all positions $j' < i$ s.t. $\chi(h, j')$ and $h \lessdot j'$, so φ holds in all children of r between $v = \tau(k)$ and $s = \tau(i)$, excluded. This is case 2a, and $\Leftarrow(\varphi, \psi)$ holds in s.

(4.11): $\chi^{\doteq}_P(\bigcirc^{\lessdot}\psi' \wedge \neg\chi^{\leqslant}_F\neg\varphi')$ holds in i. Then, there exists a position h s.t. $\chi(h, i)$ and $h \doteq i$. $\bigcirc^{\lessdot}\psi'$ holds in h, so ψ holds in node $v = \tau(h + 1)$, which is the leftmost child of $r = \tau(h)$. Since $\neg\chi^{\leqslant}_F\neg\varphi'$ holds in h, ψ' holds in all positions j' s.t. $\chi(h, j')$ and $h \lessdot j$. So, ψ holds in all children of r except (possibly) the leftmost (v) and the rightmost ($s = \tau(i)$) ones. This is case 2b, and $\Leftarrow(\varphi, \psi)$ holds in s. □

It is possible to express all POTL operators in FOL, as per Section 4.1. From this, and Lemmas 4.6, 4.7, and 4.8 together with Theorem 4.5, we derive

Theorem 4.9. *POTL = FOL with one free variable on finite OP words.*

Corollary 4.10. *The propositional operators plus* $\bigcirc^d, \ominus^d, \chi_F^d, \chi_P^d, \mathcal{U}_\chi^d, \mathcal{S}_\chi^d, \bigcirc_H^u, \ominus_H^u, \mathcal{U}_H^u, \mathcal{S}_H^u$ *are expressively complete on OP words.*

Corollary 4.11. *NWTL* $\subset$ *OPTL* $\subset$ *POTL over finite OP words.*

Corollary 4.12. *Every FO formula with at most one free variable is equivalent to one using at most three distinct variables on finite OP words.*

Corollary 4.10 follows from the definition of $\iota_\mathcal{X}$ and Theorem 4.9 (note that all other operators are shortcuts for formulas expressible with those listed). In Corollary 4.11, NWTL $\subset$ OPTL was proved in [CMP20], and OPTL $\subseteq$ POTL comes from Theorem 4.9 and the semantics of OPTL being expressible in FOL similarly to POTL, while OPTL $\subsetneq$ POTL comes from Theorem 3.3. Corollary 4.12, stating that OP words have the three-variable property, follows from the FOL semantics of POTL being expressible with just three variables.

5. First-Order Completeness on ω-Words

To prove the FO-completeness of the translation of $\mathcal{X}_{until}$ into POTL also on OP ω-words, we must prove that $\mathcal{X}_{until}$ is FO-complete on the OPM-compatible UOTs resulting from ω-words. In Section 5.1 we show that infinite OPM-compatible UOTs can be divided in two classes, depending on their shape. Then, after introducing some new notation in Section 5.2, we show how to translate a given FO formula into $\mathcal{X}_{until}$ separately for each UOT class in Sections 5.3 and 5.4, and how such translations can be combined to work on any infinite OPM-compatible UOT in Section 5.5. Our proofs exploit composition arguments on trees from [Lib09, HT87, MR99] but introduce new techniques to deal with the peculiarities of UOTs derived from ω-OPLs.

5.1. OPM-compatible ω-UOTs.

The application of function τ from Section 4 to OP ω-words results in two classes of infinite UOTs, depending on the shape of the underlying ST. In both cases, in the UOT the rightmost child of the root is not labeled with #, and nodes in the rightmost branch do not have a *right context*. τ^{-1}, which can be defined in the same way, converts such UOTs into words with open chains.

If a word w reaches infinity through right recursion, then it contains an infinite number of chains that have a left context, which we call a *pending* position, but no right context. An ωOPBA reading such a word does an infinite number of push moves, and its stack grows to infinity. The corresponding UOT $T_w = \tau(w)$ presents a single infinite branch, made of the rightmost nodes of each level. Such nodes, which we call *pending*, are in the $\lessdot$ PR when they correspond to a right recursion step, and in the $\doteq$ PR when they are siblings in the ST. Pending nodes may have left non-terminal (dot) siblings, which correspond to bodies of inner chains between two consecutive right-recursion steps. So, ω-words in which left and right recursion are alternated also fall into this class, which we call right-recursive (RR) UOTs (Figure 9).

If w reaches infinity by left recursion, then it contains an infinite number of chains sharing the same left context. An ωOPBA reading w performs an infinite sequence of pop moves, each one followed by a push, and its stack size is "ultimately bounded", i.e., the stack symbol related to such a left context remains in the stack indefinitely, and other symbols are repeatedly pushed on top of it, and popped. Thus, the stack reaches the same size infinitely many times. The rightmost branch of $T_w = \tau(w)$ ends with a node r_∞ with an infinite

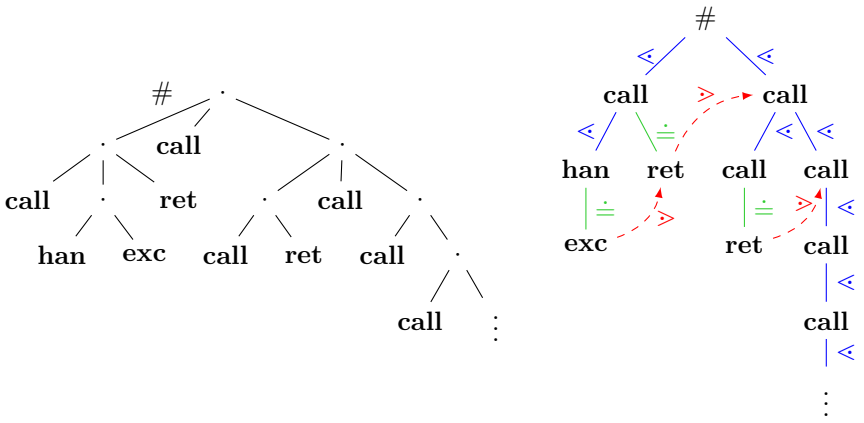

Figure 9: ST (left) and UOT (right) of the RR OP ω-word # **call han exc ret call call ret call call call** ... on OPM $M_{\mathbf{call}}$ (Figure 1).

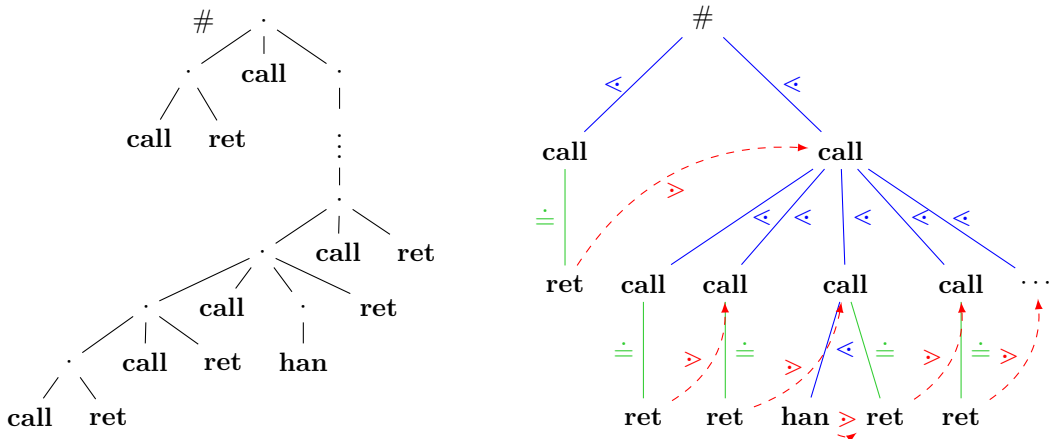

Figure 10: ST (left) and UOT (right) of the LR OP ω-word # **call ret call call ret call ret call han ret call ret** ... on OPM $M_{\mathbf{call}}$ (Figure 1).

number of children (cf. Figure 10). Node r_∞ is in the $\lessdot$ relation with all of its children, otherwise it would violate property 3 of the χ relation. This is the left-recursive (LR) class of UOTs.

An exception to the above classification may occur if the OPM is such that the transitive closure of the $\doteq$ relation is reflexive —in other words the OPM contains $\doteq$-circularities—. In this case the ST *may* contain just one node with an infinite number of children, all in the $\doteq$ PR. As a result, such nodes form *a unique infinite branch* in the corresponding UOT whose nodes are in the $\doteq$ relation unlike the case of Figure 9. This is the distinguishing feature of RR UOTs, despite the fact that in this case the stack of the ωOPBA remains "ultimately bounded" as in the case of LR UOTs. Thus, this exceptional case is attributed to the RR class.

In the following, by RR (resp. LR) word or ST we mean an OP ω-word or ST that translates to a RR (resp. LR) UOT. Next, we separately give a translation of FOL into $\mathcal{X}_{until}$ for RR and LR UOTs, and then show how to combine them to obtain completeness.

Remarks 5.1. There cannot be RR UOTs containing a node with infinitely many children, or LR UOTs where more than one node has infinite children. If this was the case, then the infinite children would appear as consecutive infinite subsets of positions in the OP ω-word isomorphic to the UOT. But this is impossible, because the set of word positions is $\mathbb{N}$, which is not dense and does not contain dense subsets.

The FO-completeness proof of the logic NWTL [AAB$^+$08] is also based on a translation of Nested Words to UOTs. However, Nested Words result in only one kind of UOT, because VPL grammars can be transformed so that words grow in only one direction. Thus, that proof does not deal with the issue of combining two separate translations.

5.2. **Notation.** Given a FO formula φ, we call its *quantifier rank* (q.r.) the maximum nesting level of its quantifiers. Let $\mathcal{M}$ be a structure on a relational signature $\langle D, R_1, \ldots, R_n\rangle$, with domain D and relations $R_1, \ldots, R_n$. The *rank-k type* of $\mathcal{M}$ is the set

$$\sigma_k(\mathcal{M}) = \{\varphi \mid \varphi \in \text{FOL}, \mathcal{M} \models \varphi \text{ and the q.r. of } \varphi \text{ is } k\},$$

while the rank-k type of $\mathcal{M}$ with a distinguished element $d \in D$ is

$$\sigma_k(\mathcal{M}, d) = \{\varphi(x) \mid \varphi(x) \in \text{FOL}, (\mathcal{M}, d) \models \varphi(x) \text{ and the q.r. of } \varphi(x) \text{ is } k\}.$$

Since the set of nonequivalent FO formulas with at most k quantifiers on a relational signature is finite, there are only finitely many rank-k types. For each rank-k type σ_k it is possible to define the Hintikka formula H_{σ_k} [Hin53], s.t. $\mathcal{M} \models H_{\sigma_k}$ iff the rank-k type of $\mathcal{M}$ is σ_k.

The compositional argument used here is based on Ehrenfeucht-Fraïssé (EF) games (see e.g., [GKL$^+$07, Imm12]) between two players, $\forall$ (the *Spoiler*) and $\exists$ (the *Duplicator*). A round of an EF game between two structures $\mathcal{M}$ and $\mathcal{M}'$ with the same signature starts with $\forall$ picking an element of the domain of either one of $\mathcal{M}$ and $\mathcal{M}'$, followed by $\exists$ answering by picking an element of the other structure. $\exists$ wins the k-round game on two structures if the map between the elements picked by $\forall$ and those picked by $\exists$ in each of the first k rounds is a partial isomorphism between $\mathcal{M}$ and $\mathcal{M}'$. We write $\mathcal{M} \sim_k \mathcal{M}'$ if $\exists$ has a winning strategy for the k-round game between $\mathcal{M}$ and $\mathcal{M}'$ and, given $a \in D$ and $a' \in D'$, we write $(\mathcal{M}, a) \sim_k (\mathcal{M}', a')$ iff $\exists$ wins the game on $\mathcal{M}$ and $\mathcal{M}'$ in whose first round $\forall$ picks a, and $\exists$ answers with a'. We write $\mathcal{M} \equiv_k \mathcal{M}'$ to state that $\mathcal{M}$ and $\mathcal{M}'$ have the same rank-k type, i.e. they satisfy exactly the same FO formulas of q.r. at most k. By the EF Theorem, $\mathcal{M} \sim_k \mathcal{M}'$ iff $\mathcal{M} \equiv_k \mathcal{M}'$, for all k.

We refer to the syntax and semantics of $\mathcal{X}_{until}$ presented in Section 4.2.2. The semantics of the until and since operators is strict, i.e. the position where they are evaluated is not part of their paths, which start with the next one. Thus, next and back operators are not needed, but we define them as shortcuts, for any $\mathcal{X}_{until}$ formula φ:

$$\bigcirc_{\Downarrow} \varphi := \Downarrow(\neg\top, \varphi) \qquad \bigcirc_{\Uparrow} \varphi := \Uparrow(\neg\top, \varphi) \qquad \bigcirc_{\Rightarrow} \varphi := \Rightarrow(\neg\top, \varphi) \qquad \bigcirc_{\Leftarrow} \varphi := \Leftarrow(\neg\top, \varphi)$$

We call the structure $\langle U, <, P\rangle$ a finite LTL word if $U \subseteq \mathbb{N}$ is finite, and an LTL ω-word if $U = \mathbb{N}$. $<$ is a linear order on U, and $P\colon AP \to \mathcal{P}(U)$ is a labeling function. The syntax of an LTL formula φ is, for a $\in AP$, $\varphi ::= \text{a} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi\,\mathcal{U}\,\varphi \mid \varphi\,\mathcal{S}\,\varphi$, where propositional operators have the usual meaning. Given an LTL word w, and a position i in it,

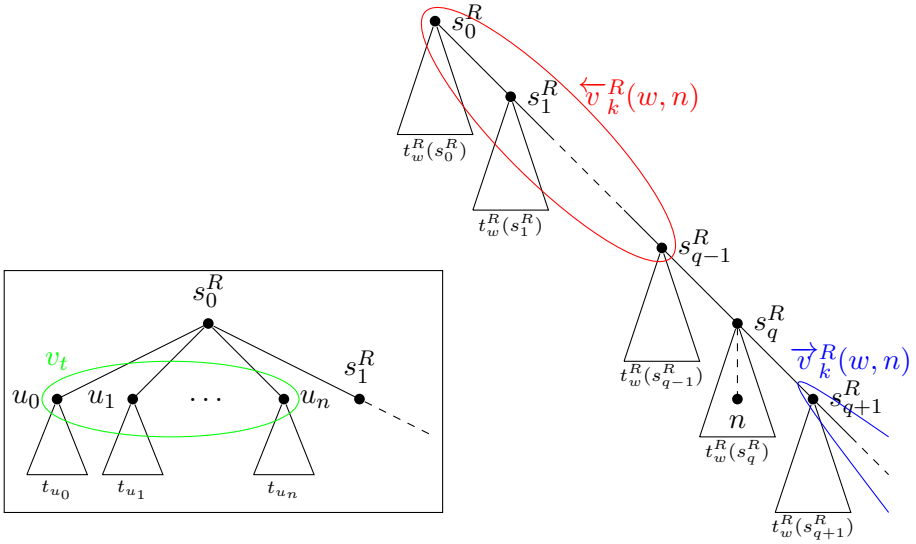

Figure 11: Parts in which we divide a RR UOT for Lemma 5.3 (right) and Lemma 5.4 (left).

- $(w,i) \models \mathrm{a}$ iff $i \in P(\mathrm{a})$;
- $(w,i) \models \psi \,\mathcal{U}\, \theta$ iff there is $j > i$ s.t. $(w,j) \models \theta$ and for any $i < j' < j$ we have $(w,j') \models \psi$;
- $(w,i) \models \psi \,\mathcal{S}\, \theta$ iff there is $j < i$ s.t. $(w,j) \models \theta$ and for any $j < j' < i$ we have $(w,j') \models \psi$.

A *future* LTL formula only contains the $\mathcal{U}$ modality, and a *past* one only contains $\mathcal{S}$.

5.3. **RR UOTs.** We prove the following:

Lemma 5.2. *Given a FO formula on UOTs $\bar{\varphi}(x)$ of q.r. $k \geq 1$, there exists a $\mathcal{X}_{until}$ formula φ_R s.t. for any OP ω-word w s.t. $T_w = \tau(w)$ is a RR UOT, and for any node $n \in T_w$ we have $(T_w, n) \models \bar{\varphi}(x)$ iff $(T_w, n) \models \varphi_R$.*

To prove Lemma 5.2, we show that φ_R can be built by combining formulas describing only parts of a UOT, as hinted in Figure 11. First, we prove that the rank-k type of such parts determines the rank-k type of the whole tree (Lemma 5.3); then, since such subdivision includes finite subtrees, we show how to express their rank-k types in $\mathcal{X}_{until}$ (Lemma 5.4); finally we use such results to translate $\bar{\varphi}(x)$.

This part of the proof partially resembles the one for the FO-completeness of NWTL in [AAB$^+$08], because of the similarity between the shape of RR UOTs and those resulting from nested ω-words. However, the two proofs diverge significantly, because Nested Words are isomorphic to binary trees, while RR UOTs are unranked.

Consider the RR UOT T_w of the statement. We denote with $s_0^R, s_1^R, \ldots$ the infinite sequence of *pending* nodes obtained by starting from the root of T_w, and always descending through the rightmost child. We call $t_w^R(s_p^R)$ the finite subtree obtained by removing the rightmost child of s_p^R and its descendants from the subtree rooted in s_p^R. If s_p^R has one single child, $t_w^R(s_p^R)$ is made of s_p^R only. Let n be any node in T_w, and let s_q^R be s.t. n is part of $t_w^R(s_q^R)$, and $s_q^R = n$ if n is a pending node. Let Γ_k be the (finite) set of all rank-k types of finite UOTs, and $\sigma_k^R(w,n) \in \Gamma_k$ be the rank-k type of $t_w^R(s_q^R)$. We define $\overleftarrow{v}_k^R(w,n)$ as a finite LTL word of length q on alphabet Γ_k, s.t. each position p, $0 \leq p \leq q-1$, is labeled

with $\sigma_k^R(w, s_p^R)$. Also, let $\overrightarrow{v}_k^R(w, n)$ be a LTL ω-word on Γ_k having each position labeled with $\sigma_k^R(w, s_{q+j+1}^R)$ for all $j \geq 0$. We give the following compositional argument:

Lemma 5.3. *Let w_1 and w_2 be two OP ω-words, such that $T_{w_1} = \tau(w_1)$ and $T_{w_2} = \tau(w_2)$ are two RR UOTs. Let i_1 and i_2 be two positions in, resp., w_1 and w_2, and let $s_1 = \tau(i_1)$ and $s_2 = \tau(i_2)$. For any $k \geq 1$, if*
(1) $\overleftarrow{v}_k^R(w_1, s_1) \equiv_k \overleftarrow{v}_k^R(w_2, s_2)$,
(2) $\overrightarrow{v}_k^R(w_1, s_1) \equiv_k \overrightarrow{v}_k^R(w_2, s_2)$ *and*
(3) $\sigma_k^R(w_1, s_1) = \sigma_k^R(w_2, s_2)$,
then $(T_{w_1}, s_1) \equiv_k (T_{w_2}, s_2)$.

Proof. By the EF Theorem, equivalences 1–2–3 imply that the respective games have a winning strategy; we refer to them by game 1, 2 and 3. We prove that $(T_{w_1}, s_1) \sim_k (T_{w_2}, s_2)$, i.e. that $\exists$ has a winning strategy in the EF game on (T_{w_1}, s_1) and (T_{w_2}, s_2) so that the thesis follows. In round 0 of the game, partial isomorphism is ensured by s_1 and s_2 having the same labels, due to hypothesis 3. In any subsequent round, suppose w.l.o.g. that $\forall$ picks a node $s^\forall$ from T_{w_1} (the converse is symmetric). Let $s_{q_1}^R$ be the pending node s.t. s_1 is part of $t_w^R(s_{q_1}^R)$, and $s_{q_2}^R$ be the pending node s.t. s_2 is part of $t_w^R(s_{q_2}^R)$. We have the following cases:

- $s^\forall = s_{q^\forall}^R$ is one of the pending nodes that are ancestors of $s_{q_1}^R$, and $q^\forall$ is the corresponding position in $\overleftarrow{v}_k^R(w_1, s_1)$. Then, $\exists$ selects $q^\exists$ in $\overleftarrow{v}_k^R(w_2, s_2)$ in response to $q^\forall$ according to her winning strategy for game 1. Her answer to $s^\forall$ is $s^\exists = s_{q^\exists}^R$ (i.e. the pending node with the same index).
- $s^\forall$ is part of a subtree $t_{w_1}^R(s_{q^\forall}^R)$ such that $s_{q^\forall}^R$ is an ancestor of $s_{q_1}^R$. Then $\exists$ chooses position $q^\exists$ in $\overleftarrow{v}_k^R(w_2, s_2)$ as before. According to game 1, $q^\forall$ and $q^\exists$ must be labeled with the same rank-k type of a subtree (i.e. $\sigma_k^R(w_1, s_{q^\forall}^R) = \sigma_k^R(w_2, s_{q^\exists}^R)$). Hence, game $t_{w_1}^R(s_{q^\forall}^R) \sim_k t_{w_2}^R(s_{q^\exists}^R)$ has a winning strategy, which $\exists$ can use to pick $s^\exists$ in $t_{w_2}^R(s_{q^\exists}^R)$.
- If the node picked by $\forall$ is the rightmost child of $s_{q_1}^R$ or one of its descendants, then $\exists$ proceeds symmetrically, but using her winning strategy on game 2.
- Finally, if $\forall$ picks a node in $t_{w_1}^R(s_{q_1}^R)$, then by 3 we have $t_{w_1}^R(s_{q_1}^R) \sim_k t_{w_2}^R(s_{q_2}^R)$, and $\exists$ answers according to her winning strategy in this game.

This strategy preserves the partial isomorphism w.r.t. the child and sibling relations and monadic predicates, as a direct consequence of rank-k type equivalences in the hypotheses. □

Lemma 5.3 shows that the rank-k type of an RR OPM-compatible UOT is determined by the rank-k types of the parts in which we divide it. Given FO formula $\bar{\varphi}(x)$, consider the set of all tuples made of (1) the rank-k type of $\overleftarrow{v}_k^R(w, s)$, (2) the rank-$k$ type of $\overrightarrow{v}_k^R(w, s)$, and (3) the type $\sigma_k^R(w, s)$, such that $(T_w, s) \models \bar{\varphi}(x)$ for any RR UOT T_w and $s \in T_w$. This set is finite, because there are only finitely many rank-k types of each component. So we can translate the formulas expressing the types in each tuple into $\mathcal{X}_{until}$ separately, and combine them to obtain one for the whole tree. Then, $\mathcal{X}_{until}$ formula φ is a disjunction of the resulting translated formulas, one for each tuple.

Before proceeding with our translation, we need to show how to express in $\mathcal{X}_{until}$ the rank-k type of a finite UOT such as $t_w^R(s_p^R)$, for some w and p, in the context of T_w. Since the rank-k type of $t_w^R(s_p^R)$ only contains information about that subtree, we need to restrict the formula expressing it to such nodes. In the following lemma we show how to do this,

thanks to a formula α^* which holds in the root of the subtree (but may hold in other parts of T_w), and allows us to restrict $\mathcal{X}_{until}$ operators so that they do not exit the subtree.

Lemma 5.4. *Let $\sigma_k(t)$, with $k \geq 1$, be the rank-k type of a finite OPM-compatible UOT t. Let r be a node of a RR or LR OPM-compatible UOT T_w with a finite number of children. Let α^* be a $\mathcal{X}_{until}$ formula that holds in r, and does not hold in subtrees rooted at children of r, except possibly the rightmost one. Then, there exists a $\mathcal{X}_{until}$ formula $\beta(t)$ that, if evaluated in r, is true iff r is the root of a subtree of T_w with rank-k type $\sigma_k(t)$, from which the subtree rooted in r's rightmost child has been erased in case α^* holds in that child.*

Proof. Let t_w be the subtree rooted at r, excluding r's rightmost child and its descendants, if α^* holds in it. We provide a formula $\beta(t)$ that holds in r iff $t_w \equiv_k t$. If t has only one node s with no children, which can be determined with a FO formula with one quantifier, then $\beta(t) := \beta_{AP}(s) \wedge \neg \bigcirc_{\Downarrow}(\neg\alpha^*)$, where $\beta_{AP}(s)$ is a Boolean combination of the atomic propositions holding in s.

Otherwise, the rank-k type of t is fully determined by

(1) the propositional symbols holding in its root s;
(2) the rank-k type of the finite LTL word v_t, whose positions $0 \leq p \leq m$ are labeled with the rank-k types of the subtrees t_{u_p} rooted at the children u_p of s (including the rightmost one).

This can be proved with a simple compositional argument.

Thus, we define

$$\beta(t) := \bigcirc_{\Downarrow}(\neg \bigcirc_{\Leftarrow} \top \wedge \beta'(t)) \wedge \beta_{AP}(s),$$

where $\beta_{AP}(s)$ is a Boolean combination of the atomic propositions holding in s, and $\beta'(t)$ characterizes $t_{u_0}, \ldots, t_{u_m}$. By this we mean that $\beta'(t)$ is such that when $\beta(t)$ holds on r, its children (except the rightmost one if α^* holds in it) must be roots of subtrees isomorphic to $t_{u_0}, \ldots, t_{u_m}$. Note that formula $\beta'(t)$ is enforced in the leftmost child (where $\bigcirc_{\Leftarrow} \top$ is false) of the node where $\beta(t)$ is evaluated.

We now show how to obtain $\beta'(t)$. Due to Kamp's Theorem [Kam68] and the separation property of LTL [GHR94], there exists a future LTL formula $\beta''(t)$ that, evaluated in the first position of v_t, completely determines its rank-k type (it can be obtained by translating into LTL the Hintikka formula equivalent to the rank-k type of v_t).

We now show how to express the rank-k types of the subtrees t_{u_p}. Since they are finite, by Marx's Theorem [Mar05, Corollary 3.3], there exists a $\mathcal{X}_{until}$ formula γ_p that, evaluated in u_p, fully determines the rank-k type of t_{u_p} (γ_p can be obtained by translating the Hintikka formula for the rank-k type of t_{u_p} into $\mathcal{X}_{until}$). Unfortunately, the separation property does not hold for $\mathcal{X}_{until}$ [BL16], and γ_p may contain $\Uparrow$ operators that, in the context of T_w, consider nodes that are not part of t_{u_p}.

There is, however, a way of syntactically transforming γ_p so that, if evaluated on a child r' of r, its paths remain constrained to $t_{r'}$, the subtree rooted in r'. Given a $\mathcal{X}_{until}$ formula ψ, it can be written as a Boolean combination of atomic propositions and until/since operators (possibly nested). We denote by $\psi^{\Downarrow}$ the formula obtained by replacing all subformulas of the form $\Uparrow(\varphi, \varphi')$, $\Rightarrow(\varphi, \varphi')$ and $\Leftarrow(\varphi, \varphi')$ at the topmost level with $\neg\top$. If γ_p is evaluated in the root of t_{u_p} outside of t, all such operators evaluate to false. So, $\gamma_p^{\Downarrow}$ in r' agrees with γ_p in the root of t_{u_p} on such subformulas. Now, take $\gamma_p^{\Downarrow}$, and recursively replace all subformulas

of the form $\Uparrow(\varphi, \varphi')$ with the following:

$$\Uparrow\Big(\varphi \wedge \neg \bigcirc_{\Uparrow} \alpha^*, (\neg \bigcirc_{\Uparrow} \alpha^* \wedge \varphi') \vee (\bigcirc_{\Uparrow} \alpha^* \wedge \varphi'^{\Downarrow})\Big).$$

We call γ'_p the obtained formula. Note that, since α^* holds in r and at most in its rightmost child, $\bigcirc_{\Uparrow} \alpha^*$ only holds in nodes u_p, $0 \leq p \leq m$. This way, we can prove that $\Uparrow$ paths in γ'_p cannot continue past u_p, and those ending in u_p depend on a formula that does not consider nodes outside the subtree rooted at u_p. Thus, when γ'_p is evaluated in u_p in the context of T_w, all its until/since operators consider exactly the same positions as the corresponding ones in γ_p evaluated outside of T_w. Hence, the two formulas are equivalent in their respective contexts, and γ'_p is true in positions that are the root of a subtree equivalent to t_{u_p}.

We now show that such transformations do not change the formula's meaning in unwanted ways. Let t_1 be a UOT, and let t_2 be a subtree of a larger UOT T, such that t_1 and t_2 are identical, and α^* holds in the parent of r_2, the root of t_2. We prove that γ'_p holds on r_2 iff γ_p holds on r_1, the root of t_1. For any subformula ψ of γ_p not at the topmost nesting level, let ψ' be the corresponding subformula in γ'_p. Then, we prove by structural induction on ψ that, for any node $s \neq r_1$ in t_1 and t_2, we have $(t_1, s) \models \psi$ iff $(T, s) \models \psi'$.

The base case, where ψ is an atomic proposition, is trivial. The composition by Boolean operators is also straightforward.

Paths considered by formulas of the form $\Rightarrow(\varphi_1, \varphi_2)$, $\Leftarrow(\varphi_1, \varphi_2)$, and $\Downarrow(\varphi_1, \varphi_2)$ cannot get out of t_2, when evaluated in one of its nodes that is not r_2. Thus, we have e.g. $(t_1, s) \models \Rightarrow(\varphi_1, \varphi_2)$ iff $(T, s) \models \Rightarrow(\varphi'_1, \varphi'_2)$ directly from the fact that $(t_1, s') \models \varphi_1$ iff $(T, s') \models \varphi'_1$ for any $s' \in t_1$ (and the same for φ_2).

Finally, let $\psi = \Uparrow(\varphi_1, \varphi_2)$. Suppose ψ is witnessed by a path in t_1 that does not include its root r_1. Then, $\neg \bigcirc_{\Uparrow} \alpha^*$ holds in all positions in such path inside t_2, and ψ' is satisfied by the same path, thanks to the inductive hypothesis. Otherwise, ψ may be witnessed by a path ending in r_1. In this case, $\neg \bigcirc_{\Uparrow} \alpha^*$ holds in all positions in the corresponding path in t_2 except the last one. There, $(\bigcirc_{\Uparrow} \alpha^* \wedge \varphi'^{\Downarrow})$ holds. In fact, φ holds in the root of t_1 where any $\Uparrow$, $\Rightarrow$ and $\Leftarrow$ operator are false, so they can correctly be replaced with $\neg\top$ in $\varphi'^{\Downarrow}$. Moreover, $\Downarrow$ operators are strict, so they must be witnessed by paths completely contained in t_1, excluding its root. Hence, by the inductive hypothesis they witness the corresponding operators in $\varphi'^{\Downarrow}$. Conversely, any path that witnesses ψ' in $s \in t_2$ must be, by construction, entirely inside t_2, so it witnesses ψ in t_1 as well. Finally, γ'_p holding on r_2 can be justified by an argument similar to the one for $\varphi'^{\Downarrow}$.

Now, we can go on with the definition of $\beta(t)$: we set $\beta'(t) := \neg\alpha^* \wedge \beta'''(t)$, where $\beta'''(t)$ is obtained from $\beta''(t)$ by

- recursively replacing all subformulas $\varphi \,\mathcal{U}\, \varphi'$ with $\Rightarrow(\neg\alpha^* \wedge \varphi, \neg\alpha^* \wedge \varphi')$;
- replacing all labels, which are rank-k types of UOTs t_{u_p}, with the corresponding γ'_p.

Thus, $\beta'(t)$ is a $\mathcal{X}_{until}$ formula that characterizes children of r, without considering the one where α^* holds (if any). □

Lemma 5.4 allows us to express in $\mathcal{X}_{until}$ the rank-k type of any subtree $t = t^R_w(s^R_p)$, for any pending node s^R_p in T_w. The root of t is a pending node, and so is its rightmost child s^R_{p+1}, which is not part of t. So, we use $\alpha^R_\infty := \neg\Uparrow(\top, \bigcirc_{\Rightarrow} \top)$, which is true in pending nodes, where the path from the current node to the root is made of rightmost nodes only. Thus, formula $\beta(t)$ from Lemma 5.4 is true in a pending node iff it is the root of a subtree equivalent to t, excluding its rightmost child.

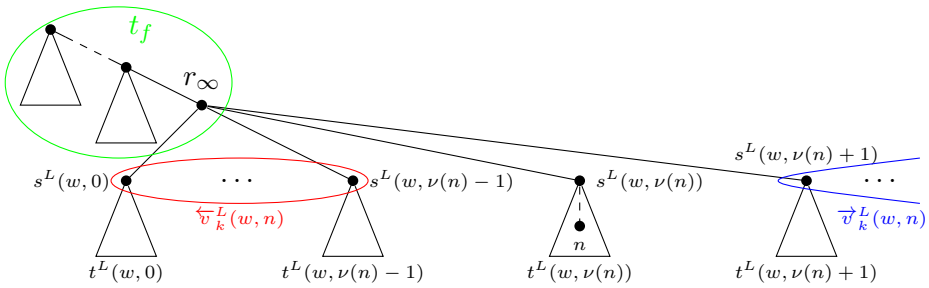

Figure 12: Parts in which we divide a LR UOT for Lemma 5.6.

Let n be the node where the translated formula is evaluated (i.e. the one corresponding to the free variable in the FO formula $\bar{\varphi}(x)$ to be translated). Let $s^R_{p_n}$ be its closest pending ancestor. According to Lemma 5.3, we need to express the rank-k types of $t^R_w(s^R_{p_n})$, $\overrightarrow{v}^R_k(w,n)$, and $\overleftarrow{v}^R_k(w,n)$.

For $t^R_w(s^R_{p_n})$, simply take formula $\beta(t^R_w(s^R_{p_n}))$.

By Kamp's Theorem and the separation property of LTL, the rank-k type of $\overrightarrow{v}^R_k(w,n)$ can be expressed by a future LTL formula $\overrightarrow{\psi}$ to be evaluated in its first position. First, we recursively take the conjunction of all subformulas of $\overrightarrow{\psi}$ with α^R_∞. Then, we recursively substitute each LTL operator $\varphi\,\mathcal{U}\,\varphi'$ with $\Downarrow(\varphi,\varphi')$, obtaining a $\mathcal{X}_{until}$ formula $\overrightarrow{\psi}'$ that evaluates its paths only on pending positions in T_w. We can prove that equivalence is kept despite such transformations by induction on the formula's structure, as we did in Lemma 5.4. Since $\overrightarrow{v}^R_k(w,n)$ is labeled with rank-k types of UOTs, we substitute any such atomic proposition $\sigma_k(t)$ in $\overrightarrow{\psi}'$ with $\beta(t)$, obtaining formula $\overrightarrow{\psi}''$, that captures the part of the tree rooted at the rightmost child of $s^R_{p_n}$.

A formula $\overleftarrow{\psi}''$ for $\overleftarrow{v}^R_k(w,n)$ can be obtained similarly, but using a past LTL formula. The Since modality can be replaced with $\Uparrow$, while other transformations remain the same.

All formulas we built so far are meant to be evaluated in a pending node. If $n = s^R_{p_n}$, then we are done. Otherwise, we evaluate in n an appropriate $\Uparrow$ formula, that can only be witnessed by a path ending in $s^R_{p_n}$. Thus, the final translation is

$$\varphi_R := (\alpha^R_\infty \wedge \varphi'_R) \vee \Uparrow\left(\neg\alpha^R_\infty, \alpha^R_\infty \wedge \varphi'_R\right)$$

where

$$\varphi'_R := \beta(t^R_w(s^R_{p_n})) \wedge \bigcirc_{\Downarrow}(\neg \bigcirc_{\Rightarrow} \top \wedge \overrightarrow{\psi}'') \wedge \bigcirc_{\Uparrow} \overleftarrow{\psi}''.$$

This concludes the proof of Lemma 5.2. □

5.4. **LR UOTs.** Let w be an OP ω-word, and $T_w = \tau(w)$ a LR UOT. We name r_∞ the node with infinite children, and denote as $t_f(w)$ the finite UOT obtained by removing all children of r_∞ from T_w. We prove the following:

Lemma 5.5. *Given a FO formula on UOTs $\bar{\varphi}(x)$ of q.r. $k \geq 1$, there are two $\mathcal{X}_{until}$ formulas φ_{L1} and φ_{L2} s.t. for any OP ω-word s.t. $T_w = \tau(w)$ is a LR UOT, and for any node $n \in T_w$ we have*

- $(T_w, n) \models \bar{\varphi}(x)$ *iff* $(T_w, n) \models \varphi_{L1}$ *when* $n \notin t_f(w)$, *and*
- $(T_w, n) \models \bar{\varphi}(x)$ *iff* $(T_w, n) \models \varphi_{L2}$ *when* $n \in t_f(w)$.

The proof is structured in a way similar to that of Lemma 5.2, but differs in the parts in which we divide the tree. In fact, we divide a LR UOT into its finite part, and two LTL words which make up the infinite children of r_∞ (see Figure 12).

We name $s^L(w,0), s^L(w,1), \dots$ the children of r_∞, and denote as $t^L(w,0), t^L(w,1), \dots$ the finite subtrees rooted at, resp., $s^L(w,0), s^L(w,1), \dots$, and $\sigma^L_k(w,0), \sigma^L_k(w,1), \dots$ their rank-k types. For all $p \geq 0$, for any node m in $t^L(w,p)$, we define the map ν so that $\nu(m) = p$. Now, let n be any node in T_w. We define $\overrightarrow{v}^L_k(w,n)$ as the LTL ω-word on the alphabet of rank-k types of finite OPM-compatible UOTs, such that each of its positions i_q, $q \geq 0$, is labeled with $\sigma^L_k(w, \nu(n)+q+1)$ if $n \notin t_f(w)$. If n is in $t_f(w)$, then each position i_q is labeled with $\sigma^L_k(w,q)$. We further define $\overleftarrow{v}^L_k(w,n)$ as the finite LTL word of length $\nu(n)$ on the same alphabet, such that each of its positions i_q, $0 \leq q \leq \nu(n)-1$, is labeled with $\sigma^L_k(w,q)$. If $\nu(n) = 0$, or n is part of $t_f(w)$, then $\overleftarrow{v}^L_k(w,n)$ is the empty word. We now prove the following composition argument:

Lemma 5.6. *Let w_1 and w_2 be two OP ω-words, such that $T_{w_1} = \tau(w_1)$ and $T_{w_2} = \tau(w_2)$ are two LR UOTs, and r^1_∞ and r^2_∞ are their nodes with infinitely many children. Let i_1 and i_2 be two positions in w_1 and w_2, such that, by letting $n_1 = \tau(i_1)$ and $n_2 = \tau(i_2)$, $n_1 \in t_f(w_1)$ iff $n_2 \in t_f(w_2)$. If*

(1) $(t_f(w_1), n_1) \equiv_k (t_f(w_2), n_2)$ *if* $n_1 \in t_f(w_1)$; $(t_f(w_1), r^1_\infty) \equiv_k (t_f(w_2), r^2_\infty)$ *otherwise;*
(2) $\overleftarrow{v}^L_k(w_1,n_1) \equiv_k \overleftarrow{v}^L_k(w_2,n_2)$;
(3) $\overrightarrow{v}^L_k(w_1,n_1) \equiv_k \overrightarrow{v}^L_k(w_2,n_2)$;
(4) $n_1 \notin t_f(w_1)$ *implies* $t^L(w_1, \nu(n_1)) \equiv_k t^L(w_2, \nu(n_2))$

then $(T_{w_1}, n_1) \equiv_k (T_{w_2}, n_2)$.

Proof. The proof is carried out by means of a standard composition argument. We give a winning strategy for $\exists$ in the k-round EF game between (T_{w_1}, n_1) and (T_{w_2}, n_2). Suppose w.l.o.g. that $\forall$ picks a node $n^\forall$ from T_{w_1}. $n^\forall$ may be part of, either, $t_f(w_1)$ or a subtree $t^L(w_1, \nu(n^\forall))$. In the former case, $\exists$ answers with a node in $t_f(w_2)$, according to her winning strategy in game 1. For the latter case, let us first suppose n_1 and n_2 are not in, resp., $t_f(w_1)$ and $t_f(w_2)$. Then, $\exists$ proceeds as follows:

(a) $\nu(n^\forall) < \nu(n_1)$. In this case, $\exists$ plays her game 2 as if $\forall$ had picked position $\nu(n^\forall)$ in $\overleftarrow{v}^L_k(w_1,n_1)$. Let $q^\exists$ be the position in $\overleftarrow{v}^L_k(w_2,n_2)$ chosen according to such strategy. Since $\overleftarrow{v}^L_k(w_1,n_1) \equiv_k \overleftarrow{v}^L_k(w_2,n_2)$ and $k \geq 1$, $\nu(n^\forall)$ and $q^\exists$ must have the same label, so we have $\sigma^L_k(w_1, \nu(n^\forall)) = \sigma^L_k(w_2, q^\exists)$. So, $t^L(w_1, \nu(n^\forall)) \equiv_k t^L(w_2, q^\exists)$, and $\exists$ may pick a node $n^\exists$ in $t^L(w_2, q^\exists)$ according to her winning strategy for the k-round game on $t^L(w_1, \nu(n^\forall))$ and $t^L(w_2, q^\exists)$, considering $n^\forall$ as $\forall$'s move.
(b) $\nu(n^\forall) = \nu(n_1)$. In this case, $\exists$ picks $n^\exists$ in T_{w_2} according to her winning strategy on the EF game for equivalence 4.
(c) $\nu(n^\forall) > \nu(n_1)$. $\exists$ may proceed as in case (a), but picking $q^\exists$ from $\overrightarrow{v}^L_k(w_2,n_2)$ according to her winning strategy on game 3.

If, instead, n_1 and n_2 are in $t_f(w_1)$ and $t_f(w_2)$, then $\overrightarrow{v}^L_k(w_1,n_1)$ (resp. $\overrightarrow{v}^L_k(w_2,n_2)$) represents all of the infinite siblings in T_{w_1} (resp. T_{w_2}), and $\overleftarrow{v}^L_k(w_1,n_1)$ and $\overleftarrow{v}^L_k(w_2,n_2)$ are the empty word. So, $\exists$ may proceed as in case (a), but picking $q^\exists$ from $\overrightarrow{v}^L_k(w_2,n_2)$ according to her winning strategy on game 3. □

We now show how to express a FO formula with one free variable with a $\mathcal{X}_{until}$ formula equivalent to it on a LR UOT. As in the RR case, we show how to represent in $\mathcal{X}_{until}$ the

rank-k types of all parts in which we divide the tree in Lemma 5.6. Let n be the node in which the $\mathcal{X}_{until}$ formula is evaluated. We need to distinguish whether $n \in t_f(w)$ or not. This is more conveniently done while also discerning such cases from the RR one. Thus, we now treat the two LR cases separately, and show how to combine them with the RR case in Section 5.5.

Suppose $n \notin t_f(w)$. Let $s_n = s^L(w, \nu(n))$ be the root of the subtree $t_n = t^L(w, \nu(n))$ containing n. Children of r_∞ can be identified by the $\mathcal{X}_{until}$ formula $\alpha^L_\infty := \bigcirc_\Rightarrow \top \wedge \neg\Rightarrow(\top, \neg \bigcirc_\Rightarrow \top)$, saying that no right sibling without right siblings is reachable from the current node (i.e. there exists no rightmost sibling). By Lemma 5.4 with $\alpha^* = \alpha^L_\infty$, there exists a $\mathcal{X}_{until}$ formula $\beta(t_n)$ that fully identifies the rank-k type of t_n if evaluated in s_n.

Moreover, $\overleftarrow{v}^L_k(w,n)$ and $\overrightarrow{v}^L_k(w,n)$ can be expressed similarly to $\overleftarrow{v}^R_k(w,n)$ and $\overrightarrow{v}^R_k(w,n)$ for RR UOTs. By Kamp's Theorem, there exists a future LTL formula $\overrightarrow{\psi}_k(w,n)$ that, evaluated in the first position of $\overrightarrow{v}^L_k(w,n)$, fully identifies its rank-k type. Recursively substitute $\varphi \,\mathcal{U}\, \varphi'$ subformulas with $\Rightarrow(\varphi, \varphi')$ in $\overrightarrow{\psi}_k(w,n)$, obtaining $\overrightarrow{\psi}'_k(w,n)$. Then, replace all rank-k types of UOTs in $\overrightarrow{\psi}'_k(w,n)$ with the respective formulas obtained from Lemma 5.4, with $\alpha^* = \alpha^L_\infty$, thus obtaining $\overrightarrow{\psi}''_k(w,n)$. Now, $\bigcirc_\Rightarrow \overrightarrow{\psi}''_k(w,n)$, evaluated in s_n, fully describes the rank-k type of all right siblings of s_n, and of the subtrees rooted in them.

Formula $\bigcirc_\Leftarrow \overleftarrow{\psi}''_k(w,n)$, which describes left siblings of s_n if evaluated in it, can be obtained symmetrically, but replacing $\mathcal{S}$ with $\Leftarrow$ in the LTL formula.

Finally, we need to describe the rank-k type of $t_f(w)$. By Marx's Theorem [Mar05, Corollary 3.3], there exists a formula ψ_{f1} that, evaluated in r_∞, describes the rank-k type of $t_f(w)$. Take the conjunction of each subformula of ψ_{f1} with $\neg\alpha^L_\infty$, and call the obtained formula ψ'_{f1}. Thus, the rank-k type of $t_f(w)$ is described by $\bigcirc_\Uparrow \psi'_{f1}$, evaluated in s_n. Finally, the rank-k type of the whole tree is described by the following formula, evaluated in n:

$$\varphi_{L1} := (\alpha^L_\infty \wedge \varphi'_{L1}) \vee \Uparrow\big(\neg\alpha^L_\infty, \alpha^L_\infty \wedge \varphi'_{L1}\big),$$

where the $\Uparrow$ operator is needed to reach s_n from n if $s_n \neq n$, and

$$\varphi'_{L1} := \bigcirc_\Uparrow \psi'_{f1} \wedge \bigcirc_\Leftarrow \overleftarrow{\psi}''_k(w,n) \wedge \bigcirc_\Rightarrow \overrightarrow{\psi}''_k(w,n) \wedge \beta(t_n).$$

Suppose, instead, $n \in t_f(w)$. Then, we express the rank-k type of $t_f(w)$ by means of a $\mathcal{X}_{until}$ formula ψ_{f2}, evaluated in n, which exists by Marx's Theorem [Mar05, Corollary 3.3]. In it, we recursively take the conjunction of subformulas with $\neg\alpha^L_\infty$, thus obtaining ψ'_{f2}.

Next, we need to describe the rank-k type of $\overrightarrow{v}^L_k(w, s^L(w,0))$ (recall that $\overleftarrow{v}^L_k(w, s^L(w,0))$ is the empty word). This can be done as in case $n \notin t_f(w)$, thus obtaining formula $\overrightarrow{\psi}''_k(w, s^L(w,0))$ which, evaluated in $s^L(w,0)$, fully identifies the children of r_∞. The latter can be identified by formula $\bigcirc_\Downarrow \alpha^L_\infty$. Thus, to describe its children from n, we use formula

$$\varphi'_{L2} := \Uparrow\Big(\top, \neg\bigcirc_\Uparrow \top \wedge \Downarrow\Big(\neg\bigcirc_\Rightarrow \top, \bigcirc_\Downarrow\big(\alpha^L_\infty \wedge \neg\bigcirc_\Leftarrow \top \wedge \overrightarrow{\psi}''_k(w, s^L(w,0))\big)\Big)\Big),$$

where the outermost $\Uparrow$ reaches the root of T_w (identified by $\neg\bigcirc_\Uparrow \top$), the inner $\Downarrow$ reaches r_∞ (identified by $\bigcirc_\Downarrow \alpha^L_\infty$) by descending through rightmost children (r_∞ is the left context of an open chain, hence a pending position), and $\neg\bigcirc_\Leftarrow \top$ identifies $s^L(w,0)$. The final formula is

$$\varphi_{L2} := \varphi'_{L2} \wedge \psi'_{f2}.$$

This concludes the proof of Lemma 5.5. □

5.5. **Synthesis.** To finish the proof, we must combine the previous cases to obtain a single $\mathcal{X}_{until}$ formula. First, note that it is possible to discern the type of UOT by means of FO formulas of q.r. at most 5. The following formula identifies RR UOTs:

$$\gamma_R := \forall x[(0R^*_{\Downarrow}x \wedge \forall y(0R^*_{\Downarrow}y \wedge yR^*_{\Downarrow}x) \implies \mu(y)) \implies \exists y(xR_{\Downarrow}y \wedge \mu(y))],$$

where $\mu(y) := \neg\exists z(yR_{\Rightarrow}z)$. γ_R means that any node x which is on the rightmost branch from the root must have a rightmost child, i.e. the rightmost branch has no end. LR UOTs are identified by the following formulas:

$$\gamma_{L1}(n) := \exists x[xR^+_{\Downarrow}n \wedge \exists y(xR_{\Downarrow}y) \wedge \forall y(xR_{\Downarrow}y \implies \exists z(yR_{\Rightarrow}z))]$$

$$\gamma_{L2}(n) := \exists x[\neg(xR^+_{\Downarrow}n) \wedge \exists y(xR_{\Downarrow}y) \wedge \forall y(xR_{\Downarrow}y \implies \exists z(yR_{\Rightarrow}z))]$$

Both $\gamma_{L1}(n)$ and $\gamma_{L2}(n)$ say there exists a node $x = r_\infty$ that has infinite children, but $\gamma_{L2}(n)$ is true iff the free variable n is in $t_f(w)$, while $\gamma_{L1}(n)$ is true otherwise.

Given a FO formula of q.r. m with one free variable $\bar{\varphi}(x)$, let $k = \max(m, 5)$. Consider the finite set $\Gamma_k = \{\sigma_k(T_w, n) \mid (T_w, n) \models \bar{\varphi}(x),\ n \in T_w\}$ of the rank-k types of OPM-compatible UOTs satisfying $\bar{\varphi}(x)$, with n as a distinguished node. For each one of them, take the corresponding Hintikka formula $H(w, n)$. Since $k \geq 5$, and the UOT type is distinguishable by formulas of q.r. at most 5, for each $\sigma_k(T_w, n)$ it is possible to tell whether it describes RR or LR UOTs. For each type, by Lemmas 5.2 and 5.5, it is possible to express $H(w, n)$ through formulas describing the rank-k types of the substructures given by the composition arguments, and translate them into $\mathcal{X}_{until}$ accordingly. Then, the translated formula φ is the XOR of one of the following formulas, for each type $\sigma_k(T_w, n) \in \Gamma_k$.

- If T_w is RR, then $\xi_R \wedge \varphi_R(\sigma_k(T_w, n))$.
- If T_w is LR, and $n \notin t_f(w)$: $\xi_L \wedge (\alpha^L_\infty \vee \Uparrow(\top, \alpha^L_\infty)) \wedge \varphi_{L1}(\sigma_k(T_w, n))$, where we assert one of the infinite siblings is reachable going upwards form n, and so n is not in $t_f(w)$.
- If T_w is LR, and $n \in t_f(w)$: $\xi_L \wedge \neg(\alpha^L_\infty \vee \Uparrow(\top, \alpha^L_\infty)) \wedge \varphi_{L2}(\sigma_k(T_w, n))$.

In the above formulas, $\varphi_R(\sigma_k(T_w, n))$, $\varphi_{L1}(\sigma_k(T_w, n))$, and $\varphi_{L2}(\sigma_k(T_w, n))$ are the formulas expressing $\sigma_k(T_w, n)$, obtained as described in the previous paragraphs. Moreover, we have

$$\xi_R := \Uparrow(\top, \neg \bigcirc_{\Uparrow} \top \wedge \neg\Downarrow(\neg \bigcirc_{\Rightarrow} \top, \neg \bigcirc_{\Rightarrow} \top \wedge \neg \bigcirc_{\Downarrow} \top)),$$

$$\xi_L := \Uparrow(\top, \neg \bigcirc_{\Uparrow} \top \wedge \Downarrow(\neg \bigcirc_{\Rightarrow} \top, \neg \bigcirc_{\Rightarrow} \top \wedge \bigcirc_{\Downarrow} \alpha^L_\infty)).$$

ξ_R identifies RR UOTs. In it, the outermost $\Uparrow$ reaches the root of the tree, and the inner $\Downarrow$ imposes that the rightmost branch has no end. ξ_L identifies LR UOTs. The outermost $\Uparrow$ also reaches the root of the tree, and the inner $\Downarrow$ is verified by a path in which all nodes are on the rightmost branch, except the last one, which is one of the infinite siblings.

Boolean queries can be expressed as Boolean combinations of formulas of the form $\bar{\varphi} := \exists x(\bar{\varphi}'(x))$. Then, it is possible to translate $\bar{\varphi}'(x)$ as above, to obtain $\mathcal{X}_{until}$ formula φ', and $\Downarrow(\top, \varphi')$ is such that $(T_w, 0) \models \Downarrow(\top, \varphi')$ iff $T_w \models \bar{\varphi}$, for any OPM-compatible UOT T_w.

Thus, we can state

Theorem 5.7. *$\mathcal{X}_{until}$ = FOL with one free variable on OPM-compatible ω-UOTs.*

Thanks to Theorem 5.7, we can extend the results entailed by the translation of Section 4.2.2 to ω-words.

Theorem 5.8. *POTL = FOL with one free variable on OP ω-words.*

Corollary 5.9. *The propositional operators plus $\bigcirc^d, \ominus^d, \chi^d_F, \chi^d_P, \mathcal{U}^d_\chi, \mathcal{S}^d_\chi, \bigcirc^u_H, \ominus^u_H, \mathcal{U}^u_H, \mathcal{S}^u_H$ are expressively complete on OP ω-words.*

The containment relations in Corollary 4.11 are easily extended to ω-words:

Corollary 5.10. *NWTL $\subset$ OPTL $\subset$ POTL over OP ω-words.*

Moreover,

Corollary 5.11. *Every FO formula with at most one free variable is equivalent to one using at most three distinct variables on OP ω-words.*

6. Decidability, Satisfiability, and Model Checking

The decidability of POTL is a consequence of its being equivalent to the FO fragment of the MSO characterization of OPLs [LMPP15]. More practical algorithms for satisfiability and model checking can be obtained by building OPAs equivalent to POTL formulas. Such construction procedure, which is quite involved, is detailed in [CMP21]. Thus,

Theorem 6.1 ([CMP21])**.** *Given a POTL formula φ, we can build an OPA (or an ωOPBA) $\mathcal{A}_\varphi$ of size $2^{O(|\varphi|)}$ accepting the language defined by φ.*

Emptiness of OPA and ωOPBA is decidable in polynomial time by adapting techniques originally developed for Pushdown Systems and Recursive State Machines, such as *saturation* [ABE18], or graph-theoretic algorithms [ACEM05]. Hence, satisfiability of a formula φ can be tested in time exponential in $|\varphi|$ by building $\mathcal{A}_\varphi$ and checking its emptiness. By Corollaries 4.11 and 5.10, we can extend the EXPTIME-hardness result for NWTL [AAB$^+$08] to POTL, obtaining

Corollary 6.2. *POTL satisfiability is EXPTIME-complete.*

Since there are practical algorithms for computing the intersection of both OPA and ωOPBA [LMPP15], model checking of OPA (or ωOPBA) models against a formula φ can be done by building $\mathcal{A}_{\neg\varphi}$, and checking emptiness of their intersection. Thus,

Corollary 6.3. *POTL model checking against OPA and ωOPBA is EXPTIME-complete.*

An experimental evaluation of the POTL model-checking procedure is reported in [CMP21]. Just to give an idea of its practicality, we report that formula

$$\Box\big((\mathbf{call} \wedge \mathrm{p}_B \wedge \mathit{Scall}(\top, \mathrm{p}_A)) \implies \mathit{CallThr}(\top)\big)$$

from Section 3.3 has been successfully checked against the OPA of Figure 4 in just 867 ms, with a RAM occupancy of 70 MiB, on a laptop with a 2.2 GHz Intel processor and 15 GiB of RAM, running Ubuntu GNU/Linux 20.04.

7. Conclusions

We introduced the temporal logic POTL, and proved its equivalence to FOL on both OP finite and ω-words. Thus, thanks to an independent result [MPC20a], the languages defined through POTL formulas coincide with the class of aperiodic OPLs. We also proved that POTL is strictly more expressive than temporal logics with explicit context-free-aware modalities in the literature, including OPTL, which is also based on OPLs. Such proofs are technically quite involved, which is unsurprising, given the difficulties encountered in analogous problems, even if based on the simpler framework of Nested Words [AAB$^+$08] (recall that the relationship between CaRet and NWTL remains unknown). The same

hardships, however, do not afflict the algorithmic complexity of satisfiability and model checking, which are not higher than those of nested-words logics. Thus, we argue that the strong gain in expressive power w.r.t. previous approaches to model checking CFLs brought by POTL is worth the technicalities needed to achieve the present —and future— results.

On the other hand, POTL shows promising results in its applications: [CMP21] reports on a complete model-checker for POTL and on the first encouraging experiments on a benchmark of practical interest.

In our view, POTL is the theoretical foundation on top of which to build a complete, practical and user-friendly environment to specify and verify properties of many pushdown-based systems.

While logics such as CaRet and NWTL can be viewed as the extension of LTL to Nested Words, POTL can be seen as the extension of LTL to OPLs and OP words. An interesting path for future investigations is the extension of branching-time logics such as CTL to OPLs. Something similar has been done for Nested Words in [ACM11], where a μ-calculus of Nested Trees is introduced. Nested-words logics have also been augmented in other directions: [BS14] introduces a temporal logic capturing the whole class of VPLs, while timed extensions of CaRet are given in [BMP18]. The same extensions could be attempted for POTL too.

References

[AAB+08] Rajeev Alur, Marcelo Arenas, Pablo Barceló, Kousha Etessami, Neil Immerman, and Leonid Libkin. First-order and temporal logics for nested words. *LMCS*, 4(4), 2008. doi:10.2168/LMCS-4(4:11)2008.

[ABE+05] Rajeev Alur, Michael Benedikt, Kousha Etessami, Patrice Godefroid, Thomas Reps, and Mihalis Yannakakis. Analysis of recursive state machines. *ACM Trans. Program. Lang. Syst.*, 27(4):786–818, 2005. doi:10.1145/1075382.1075387.

[ABE18] Rajeev Alur, Ahmed Bouajjani, and Javier Esparza. Model checking procedural programs. In *Handbook of Model Checking*, pages 541–572. Springer, 2018. doi:10.1007/978-3-319-10575-8_17.

[Abr00] David Abrahams. Exception-Safety in Generic Components. In *Generic Programming*, pages 69–79. Springer, 2000. doi:10.1007/3-540-39953-4_6.

[ACEM05] Rajeev Alur, Swarat Chaudhuri, Kousha Etessami, and Parthasarathy Madhusudan. On-the-fly reachability and cycle detection for recursive state machines. In *TACAS 2005*, volume 3440 of *LNCS*, pages 61–76. Springer, 2005. doi:10.1007/978-3-540-31980-1_5.

[ACM11] Rajeev Alur, Swarat Chaudhuri, and Parthasarathy Madhusudan. Software model checking using languages of nested trees. *ACM Trans. Program. Lang. Syst.*, 33(5):15:1–15:45, 2011. doi:10.1145/2039346.2039347.

[AEM04] Rajeev Alur, Kousha Etessami, and Parthasarathy Madhusudan. A temporal logic of nested calls and returns. In *TACAS 2004*, pages 467–481. Springer, 2004. doi:10.1007/978-3-540-24730-2_35.

[AF16] Rajeev Alur and Dana Fisman. Colored nested words. In *LATA 2016*, volume 9618 of *LNCS*, pages 143–155. Springer, 2016. doi:10.1007/978-3-319-30000-9_11.

[AM04] Rajeev Alur and Parthasarathy Madhusudan. Visibly pushdown languages. In *ACM STOC*, pages 202–211, 2004. doi:10.1145/1007352.1007390.

[AM09] Rajeev Alur and Parthasarathy Madhusudan. Adding nesting structure to words. *JACM*, 56(3), 2009. doi:10.1145/1516512.1516518.

[BCM+15] Alessandro Barenghi, Stefano Crespi Reghizzi, Dino Mandrioli, Federica Panella, and Matteo Pradella. Parallel parsing made practical. *Sci. Comput. Program.*, 112:195–226, 2015. doi:10.1016/j.scico.2015.09.002.

[BEM97] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR '97*, pages 135–150. Springer, 1997. doi:10.1007/3-540-63141-0_10.

[BH96] Ahmed Bouajjani and Peter Habermehl. Constrained properties, semilinear systems, and Petri nets. In *CONCUR'96*, volume 1119 of *LNCS*, pages 481–497. Springer, 1996. doi:10.1007/3-540-61604-7_71.

[BL16] Michael Benedikt and Clemens Ley. Limiting until in ordered tree query languages. *ACM Trans. Comput. Log.*, 17(2):14:1–14:34, 2016. doi:10.1145/2856104.

[BMP18] Laura Bozzelli, Aniello Murano, and Adriano Peron. Timed context-free temporal logics. In *GandALF 2018*, volume 277 of *EPTCS*, pages 235–249. Open Publishing Association, 2018. doi:10.4204/EPTCS.277.17.

[BR00] Thomas Ball and Sriram K. Rajamani. Bebop: A symbolic model checker for boolean programs. In *SPIN Model Checking and Software Verification*, pages 113–130. Springer, 2000. doi:10.1007/10722468_7.

[BS14] Laura Bozzelli and César Sánchez. Visibly linear temporal logic. In *Automated Reasoning*, pages 418–433. Springer, 2014. doi:10.1007/978-3-319-08587-6_33.

[CM12] Stefano Crespi Reghizzi and Dino Mandrioli. Operator Precedence and the Visibly Pushdown Property. *JCSS*, 78(6):1837–1867, 2012. doi:10.1016/j.jcss.2011.12.006.

[CMM78] Stefano Crespi Reghizzi, Dino Mandrioli, and Daniel F. Martin. Algebraic Properties of Operator Precedence Languages. *Information and Control*, 37(2):115–133, May 1978. doi:10.1016/S0019-9958(78)90474-6.

[CMP20] Michele Chiari, Dino Mandrioli, and Matteo Pradella. Operator precedence temporal logic and model checking. *Theor. Comput. Sci.*, 848:47–81, 2020. doi:10.1016/j.tcs.2020.08.034.

[CMP21] Michele Chiari, Dino Mandrioli, and Matteo Pradella. Model-checking structured context-free languages. In *CAV '21*, volume 12760 of *LNCS*, pages 387—410. Springer, 2021. doi:10.1007/978-3-030-81688-9_18.

[EHRS00] Javier Esparza, David Hansel, Peter Rossmanith, and Stefan Schwoon. Efficient algorithms for model checking pushdown systems. In *CAV 2000*, volume 1855 of *LNCS*, pages 232–247. Springer, 2000. doi:10.1007/10722167_20.

[EKS03] Javier Esparza, Antonín Kučera, and Stefan Schwoon. Model checking LTL with regular valuations for pushdown systems. *Information and Computation*, 186(2):355–376, 2003. doi:10.1016/S0890-5401(03)00139-1.

[Eme90] E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 995–1072. Elsevier, 1990. doi:10.1016/B978-0-444-88074-1.50021-4.

[Flo63] Robert W. Floyd. Syntactic Analysis and Operator Precedence. *JACM*, 10(3):316–333, 1963. doi:10.1145/321172.321179.

[Gab81] Dov M. Gabbay. Expressive functional completeness in tense logic. In *Aspects of Philosophical Logic*, pages 91–117. Springer, 1981. doi:10.1007/978-94-009-8384-7_4.

[GHR94] Dov M. Gabbay, Ian Hodkinson, and Mark Reynolds. *Temporal Logic: Mathematical Foundations and Computational Aspects*. Oxford University Press, New York, NY, USA, 1994.

[GJ08] Dick Grune and Ceriel J. Jacobs. *Parsing techniques: a practical guide*. Springer, New York, 2008. doi:10.1007/978-0-387-68954-8.

[GKL+07] Erich Grädel, Phokion G. Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Y. Vardi, Yde Venema, and Scott Weinstein. *Finite Model Theory and its applications*. Springer, 2007. doi:10.1007/3-540-68804-8.

[Har78] Michael A. Harrison. *Introduction to Formal Language Theory*. Addison Wesley, 1978.

[Heu91] Uschi Heuter. First-order properties of trees, star-free expressions, and aperiodicity. *ITA*, 25:125–145, 1991. doi:10.1051/ita/1991250201251.

[Hin53] Jaakko Hintikka. Distributive normal forms in the calculus of predicates. *Acta Philosophica Fennica*, 6:71, 1953.

[HKT02] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*, pages 99–217. Springer, 2002. doi:10.1007/978-94-017-0456-4_2.

[HT87] Thilo Hafer and Wolfgang Thomas. Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree. In *ICALP'87*, volume 267 of *LNCS*, pages 269–279, Berlin, Heidelberg, 1987. Springer. doi:10.1007/3-540-18088-5_22.

[Imm12] Neil Immerman. *Descriptive complexity*. Springer, 2012.

[JLT99] Thomas Jensen, Daniel Le Metayer, and Tommy Thorn. Verification of control flow based security properties. In *Proc. '99 IEEE Symp. on Security and Privacy*, pages 89–103, 1999. doi:10.1109/SECPRI.1999.766902.

[JPR18] Ranjit Jhala, Andreas Podelski, and Andrey Rybalchenko. Predicate abstraction for program verification. In *Handbook of Model Checking*, pages 447–491. Springer, 2018. doi:10.1007/978-3-319-10575-8_15.

[Kam68] Hans Kamp. *Tense logic and the theory of linear order*. PhD thesis, University of California, Los Angeles, 1968.

[KPV02] Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. Pushdown Specifications. In *LPAR 2002*, volume 2514 of *LNCS*, pages 262–277. Springer, 2002. doi:10.1007/3-540-36078-6_18.

[Lib09] Leonid Libkin. The finite model theory toolbox of a database theoretician. In *PODS '09*, pages 65–76, New York, NY, USA, 2009. ACM. doi:10.1145/1559795.1559807.

[LMPP15] Violetta Lonati, Dino Mandrioli, Federica Panella, and Matteo Pradella. Operator precedence languages: Their automata-theoretic and logic characterization. *SIAM J. Comput.*, 44(4):1026–1088, 2015. doi:10.1137/140978818.

[LS10] Leonid Libkin and Cristina Sirangelo. Reasoning about XML with temporal logics and automata. *J. Appl. Log.*, 8(2):210–232, 2010. doi:10.1016/j.jal.2009.09.005.

[LST94] Clemens Lautemann, Thomas Schwentick, and Denis Thérien. Logics for context-free languages. In *CSL '94*, pages 205–216, 1994. doi:10.1007/BFb0022257.

[Mar04] Maarten Marx. Conditional XPath, the first order complete XPath dialect. In *PODS '04*, page 13, New York, USA, 2004. ACM Press. doi:10.1145/1055558.1055562.

[Mar05] Maarten Marx. Conditional XPath. *ACM Trans. Database Syst.*, 30(4):929–959, 2005. doi:10.1145/1114244.1114247.

[McN67] Robert McNaughton. Parenthesis Grammars. *JACM*, 14(3):490–500, 1967. doi:10.1145/321406.321411.

[Meh80] Kurt Mehlhorn. Pebbling mountain ranges and its application of DCFL-recognition. In *ICALP '80*, volume 85 of *LNCS*, pages 422–435, 1980. doi:10.1007/3-540-10003-2_89.

[MP71] Robert McNaughton and Seymour Papert. *Counter-free Automata*. MIT Press, Cambridge, USA, 1971.

[MP18] Dino Mandrioli and Matteo Pradella. Generalizing input-driven languages: Theoretical and practical benefits. *Computer Science Review*, 27:61–87, 2018. doi:10.1016/j.cosrev.2017.12.001.

[MPC20a] Dino Mandrioli, Matteo Pradella, and Stefano Crespi Reghizzi. Aperiodicity, star-freeness, and first-order definability of structured context-free languages. *CoRR*, abs/2006.01236, 2020. arXiv:2006.01236.

[MPC20b] Dino Mandrioli, Matteo Pradella, and Stefano Crespi Reghizzi. Star-freeness, first-order definability and aperiodicity of structured context-free languages. In *ICTAC 2020*, volume 12545, pages 161–180, Macau, China, November 2020. Springer. doi:10.1007/978-3-030-64276-1_9.

[MR99] Faron Moller and Alexander Moshe Rabinovich. On the expressive power of CTL*. In *LICS '99*, pages 360–368, Trento, Italy, 1999. IEEE Computer Society. doi:10.1109/LICS.1999.782631.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *FOCS '77*, pages 46–57, Providence, Rhode Island, USA, 1977. IEEE Computer Society. doi:10.1109/SFCS.1977.32.

[Tha67] James W. Thatcher. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *Journ. of Comp. and Syst.Sc.*, 1:317–322, 1967. doi:10.1016/S0022-0000(67)80022-9.

[Tho84] Wolfgang Thomas. Logical aspects in the study of tree languages. In *CAAP'84, 9th Colloquium on Trees in Algebra and Programming, Bordeaux, France, March 5-7, 1984, Proceedings*, pages 31–50. Cambridge University Press, 1984.

Appendix A. Omitted Proofs: Properties of the χ Relation

In the following lemma, we prove the properties of the chain relation.

Lemma A.1. *Given an OP word w and positions i, j in it, the following properties hold.*

(1) *If $\chi(i,j)$ and $\chi(h,k)$, for any h,k in w, then we have $i < h < j \implies k \leq j$ and $i < k < j \implies i \leq h$.*
(2) *If $\chi(i,j)$, then $i \lessdot i+1$ and $j-1 \gtrdot j$.*
(3) *Consider all positions (if any) $i_1 < i_2 < \cdots < i_n$ s.t. $\chi(i_p, j)$ for all $1 \leq p \leq n$. We have $i_1 \lessdot j$ or $i_1 \doteq j$ and, if $n > 1$, $i_q \gtrdot j$ for all $2 \leq q \leq n$.*
(4) *Consider all positions (if any) $j_1 < j_2 < \cdots < j_n$ s.t. $\chi(i, j_p)$ for all $1 \leq p \leq n$. We have $i \gtrdot j_n$ or $i \doteq j_n$ and, if $n > 1$, $i \lessdot j_q$ for all $1 \leq q \leq n-1$.*

Proof. In the following, we denote by c_p the character labeling word position p, and by writing ${}^{c_{-1}}[x_0 c_0 x_1 \dots x_n c_n x_{n+1}]^{c_{n+1}}$ we imply c_{-1} and c_{n+1} are the context of a simple or composed chain, in which either $x_p = \varepsilon$, or ${}^{c_{p-1}}[x_p]^{c_p}$ is a chain, for each p.

(1) Suppose, by contradiction, that $\chi(i,j)$, $\chi(h,k)$, and $i < h < j$, but $k > j$. Consider the case in which $\chi(i,j)$ is the innermost chain whose body contains h, so it is of the form ${}^{c_i}[x_0 c_0 \dots c_h x_p c_p \dots c_n x_{n+1}]^{c_j}$ or ${}^{c_i}[x_0 c_0 \dots c_h x_{n+1}]^{c_j}$. By the definition of chain, we have either $c_h \doteq c_p$ or $c_h \gtrdot c_j$, respectively.

    Since $\chi(h,k)$, this chain must be of the form ${}^{c_h}[x_p c_p \dots]^{c_k}$ or ${}^{c_h}[x_{n+1} c_j \dots]^{c_k}$, implying $c_h \lessdot c_p$ or $c_h \lessdot c_j$, respectively. This means there is a conflict in the OPM, contradicting the hypothesis that w is an OP word.

    In case $\chi(i,j)$ is not the innermost chain whose body contains h, we can reach the same contradiction by inductively considering the chain between i and j containing h in its body. Moreover, it is possible to reach a symmetric contradiction with the hypothesis $\chi(i,j)$, $\chi(h,k)$, and $i < k < j$, but $i > h$.
(2) Trivially follows from the definition of chain.
(3) We prove that only i_1 can be s.t. $i_1 \lessdot j$ or $i_1 \doteq j$. Suppose, by contradiction, that for some $r > 1$ we have $i_r \lessdot j$ or $i_r \doteq j$.

    If $i_r \lessdot j$, by the definition of chain, j must be part of the body of another composed chain whose left context is i_r. So, w contains a structure of the form ${}^{c_{i_r}}[x_0 c_j \dots]^{c_k}$ where $|x_0| \geq 1$, ${}^{c_{i_r}}[x_0]^{c_j}$, and $k > j$ is s.t. $\chi(i_r, k)$. This contradicts the hypothesis that $\chi(i_1, j)$, because such a chain would cross $\chi(i_r, k)$, contradicting property (1).

    If $i_r \doteq j$, then w contains a structure ${}^{c_{i_{r-1}}}[\dots c_{i_r} x_{i_r}]^{c_j}$, with $|x_{i_r}| \geq 1$ and ${}^{c_{i_r}}[x_{i_r}]^{c_j}$. By the definition of chain, we have $i_r \gtrdot j$, which contradicts the hypothesis.

    Thus, the only remaining alternative for $r > 1$ is $i_r \gtrdot j$.

    Similarly, if we had $i_1 \gtrdot j$, the definition of chain would lead to the existence of a position $h < i_1$ s.t. $\chi(h,j)$, which contradicts the hypothesis that i_1 is the leftmost of such positions. $i_1 \lessdot j$ and $i_1 \doteq j$ do not lead to such contradictions.
(4) The proof is symmetric to the previous one. □

Appendix B. Omitted Proofs: Expansion Laws

We prove the following expansion laws for POTL:

$$\varphi\,\mathcal{U}^t_\chi\,\psi \equiv \psi \vee \Big(\varphi \wedge \big(\bigcirc^t(\varphi\,\mathcal{U}^t_\chi\,\psi) \vee \chi^t_F(\varphi\,\mathcal{U}^t_\chi\,\psi)\big)\Big) \tag{B.1}$$

$$\varphi\,\mathcal{S}^t_\chi\,\psi \equiv \psi \vee \Big(\varphi \wedge \big(\ominus^t(\varphi\,\mathcal{S}^t_\chi\,\psi) \vee \chi^t_P(\varphi\,\mathcal{S}^t_\chi\,\psi)\big)\Big) \tag{B.2}$$

$$\varphi\,\mathcal{U}^u_H\,\psi \equiv (\psi \wedge \chi^d_P\top \wedge \neg\chi^u_P\top) \vee \big(\varphi \wedge \bigcirc^u_H(\varphi\,\mathcal{U}^u_H\,\psi)\big) \tag{B.3}$$

$$\varphi\,\mathcal{S}^u_H\,\psi \equiv (\psi \wedge \chi^d_P\top \wedge \neg\chi^u_P\top) \vee \big(\varphi \wedge \ominus^u_H(\varphi\,\mathcal{S}^u_H\,\psi)\big) \tag{B.4}$$

$$\varphi\,\mathcal{U}^d_H\,\psi \equiv (\psi \wedge \chi^u_F\top \wedge \neg\chi^d_F\top) \vee \big(\varphi \wedge \bigcirc^d_H(\varphi\,\mathcal{U}^d_H\,\psi)\big) \tag{B.5}$$

$$\varphi\,\mathcal{S}^d_H\,\psi \equiv (\psi \wedge \chi^u_F\top \wedge \neg\chi^d_F\top) \vee \big(\varphi \wedge \ominus^d_H(\varphi\,\mathcal{S}^d_H\,\psi)\big) \tag{B.6}$$

Lemma B.1. *Given a word w on an OP alphabet $(\mathcal{P}(AP), M_{AP})$, two POTL formulas φ and ψ, for any position i in w the following equivalence holds:*

$$\varphi\,\mathcal{U}^d_\chi\,\psi \equiv \psi \vee \Big(\varphi \wedge \big(\bigcirc^d(\varphi\,\mathcal{U}^d_\chi\,\psi) \vee \chi^d_F(\varphi\,\mathcal{U}^d_\chi\,\psi)\big)\Big).$$

Proof. **[Only if]** Suppose $\varphi\,\mathcal{U}^d_\chi\,\psi$ holds in i. If ψ holds in i, the equivalence is trivially verified. Otherwise, $\varphi\,\mathcal{U}^d_\chi\,\psi$ is verified by a DSP $i = i_0 < i_1 < \cdots < i_n = j$ with $n \geq 1$, s.t. $(w, i_p) \models \varphi$ for $0 \leq p < n$ and $(w, i_n) \models \psi$. Note that, by the definition of DSP, any suffix of that path is also a DSP ending in j. Consider position i_1: φ holds in it, and it can be either

- $i_1 = i + 1$. Then either $i \lessdot (i+1)$ or $i \doteq (i+1)$, and path $i_1 < i_2 < \cdots < i_n = j$ is the DSP between i_1 and j, and φ holds in all i_p with $1 \leq p < n$, and ψ in j_n. So, $\varphi\,\mathcal{U}^d_\chi\,\psi$ holds in i_1, and $\bigcirc^d(\varphi\,\mathcal{U}^d_\chi\,\psi)$ holds in i.
- $i_1 > i + 1$. Then, $\chi(i, i_1)$, and $i \lessdot i_1$ or $i \doteq i_1$. Since $i_1 < i_2 < \cdots < i_n = j$ is the DSP from i_1 to j, $\varphi\,\mathcal{U}^d_\chi\,\psi$ holds in i_1, and so does $\chi^d_F(\varphi\,\mathcal{U}^d_\chi\,\psi)$ in i.

[If] Suppose the right-hand side of the equivalence holds in i. The case $(w, i) \models \psi$ is trivial, so suppose ψ does not hold in i. Then φ holds in i, and either:

- $\bigcirc^d(\varphi\,\mathcal{U}^d_\chi\,\psi)$ holds in i. Then, we have $i \lessdot (i+1)$ or $i \doteq (i+1)$, and there is a DSP $i + 1 = i_1 < i_2 < \cdots < i_n = j$, with φ holding in all i_p with $1 \leq p < n$, and ψ in i_n.
  - If $i \doteq (i+1)$, it is not the left context of any chain, and $i = i_0 < i_1 < i_2 < \cdots < i_n$ is a DSP satisfying $\varphi\,\mathcal{U}^d_\chi\,\psi$ in i.
  - Otherwise, let $k = \min\{h \mid \chi(i, h)\}$: we have $k > j$, because a DSP cannot cross right chain contexts. So, adding i to the DSP generates another DSP, because there is no position h s.t. $\chi(i, h)$ with $h \leq j$, and the successor of i in the path can only be $i_1 = i+1$.
- $\chi^d_F(\varphi\,\mathcal{U}^d_\chi\,\psi)$ holds in i. Then, there exists a position k s.t. $\chi(i, k)$ and $i \lessdot k$ or $i \doteq k$ and $\varphi\,\mathcal{U}^d_\chi\,\psi$ holds in k, because of a DSP $k = i_1 < i_2 < \cdots < i_n = j$. If $k = \max\{h \mid h \leq j \wedge \chi(i, h) \wedge (i \lessdot h \vee i \doteq h)\}$, then $i = i_0 < i_1 < i_2 < \cdots < i_n$ is a DSP by definition, and since φ holds in i, $\varphi\mathcal{U}^d_\chi\psi$ is satisfied in it. Otherwise, let $k' = \max\{h \mid h \leq j \wedge \chi(i, h) \wedge (i \lessdot h \vee i \doteq h)\}$. Since $i_1 > i$ and chains cannot cross, there exists a value q, $1 < q \leq n$, s.t. $i_q = k'$. Thus $i_q < i_{q+1} < \cdots < i_n = j$ is a DSP, so $\varphi\,\mathcal{U}^d_\chi\,\psi$ holds in i_q too. The path $i < i_q < \cdots < i_n$ is a DSP, and $\varphi\,\mathcal{U}^d_\chi\,\psi$ holds in i. □

The proofs for the summary since and upward summary until operators are symmetric.

Lemma B.2. *Given a word w on an OP alphabet $(\mathcal{P}(AP), M_{AP})$, and two POTL formulas φ and ψ, for any position i in w the following equivalence holds:*

$$\varphi \,\mathcal{U}^u_H\, \psi \equiv (\psi \wedge \chi^d_P \top \wedge \neg \chi^u_P \top) \vee \big(\varphi \wedge \bigcirc^u_H (\varphi \,\mathcal{U}^u_H\, \psi)\big).$$

Proof. **[Only if]** Suppose $\varphi \mathcal{U}^u_H \psi$ holds in i. Then, there exists a path $i = i_0 < i_1 < \cdots < i_n$, $n \geq 0$, and a position $h < i$ s.t. $\chi(h, i_p)$ and $h \lessdot i_p$ for each $0 \leq p \leq n$, φ holds in all i_q for $0 \leq q < n$, and ψ holds in i_n. If $n = 0$, ψ holds in $i = i_0$, and so does $\chi^d_P \top$, but $\chi^u_P \top$ does not. Otherwise, the path $i_1 < \cdots < i_n$ is also a UHP, so $\varphi \,\mathcal{U}^u_H\, \psi$ is true in i_1. Therefore, φ holds in i, and so does $\bigcirc^u_H (\varphi \,\mathcal{U}^u_H\, \psi)$.

[If] If $\chi^d_P \top$ holds in i but $\chi^u_P \top$ does not, then there exists a position $h < i$ s.t. $\chi(h, i)$ and $h \lessdot i$. If ψ also holds in i, then $\varphi \mathcal{U}^u_H \psi$ is trivially satisfied in i by the path made of only i itself. Otherwise, if $\bigcirc^u_H (\varphi \,\mathcal{U}^u_H\, \psi)$ holds in i, then there exist a position $h < i$ s.t. $\chi(h, i)$ and $h \lessdot i$, and a position i_1 which is the minimum one s.t. $i_1 > i$, $\chi(h, i_1)$ and $h \lessdot i_1$. In i_1, $\varphi \,\mathcal{U}^u_H\, \psi$ holds, so it is the first position of a UHP $i_1 < i_2 < \cdots < i_n$. Since φ also holds in i, the path $i = i_0 < i_1 < \cdots < i_n$ is also a UHP, satisfying $\varphi \,\mathcal{U}^u_H\, \psi$ in i. □

The proofs for the other hierarchical operators are analogous.