Temporal Logic and Model Checking for Operator Precedence Languages*

Michele Chiari Dino Mandrioli Matteo Pradella†
DEIB, Politecnico di Milano
P.zza L. Da Vinci, 32, 20133, Milano, Italy
michele.chiari@mail.polimi.it dino.mandrioli@polimi.it matteo.pradella@polimi.it

In the last decades much research effort has been devoted to extending the success of model checking from the traditional field of finite state machines and various versions of temporal logics to suitable subclasses of context-free languages and appropriate extensions of temporal logics. To the best of our knowledge such attempts only covered *structured languages*, i.e. languages whose structure is immediately "visible" in their sentences, such as tree-languages or visibly pushdown ones. In this paper we present a new temporal logic suitable to express and automatically verify properties of *operator precedence languages*. This "historical" language family has been recently proved to enjoy fundamental algebraic and logic properties that make it suitable for model checking applications yet breaking the barrier of visible-structure languages (in fact the original motivation of its inventor Floyd was just to support efficient *parsing*, i.e. building the "hidden syntax tree" of language sentences). We prove that our logic is at least as expressive as analogous logics defined for visible pushdown languages yet covering a much more powerful family; we design a procedure that, given a formula in our logic builds an automaton recognizing the sentences satisfying the formula, whose size is at most exponential in the length of the formula.

Keywords: Operator Precedence Languages, Visibly Pushdown Languages, Input Driven Languages, Linear Temporal Logic, Model Checking.

1 Introduction

Since the pioneering works by Floyd, Hoare, McNaughton, Büchi and many others, the investigation of the relation between formal language and automata theory and mathematical logic has been an exciting and productive research field, whose main perspective and goal was the *formal correctness verification*, i.e., a mathematical proof that a given design, formalized as a suitable abstract machine, guarantees system requirements, formalized in terms of mathematical logic formulas. Whereas the early work by Floyd, Hoare, Dijkstra and others pursued the full generality of Turing complete computational formalisms, such as normal programming languages, and consequently made the verification problem undecidable and dependent on human inspection and skill, the independent approach by Büchi, McNaughton and others focused on the restricted but practically quite relevant families of finite state machines (FSMs) on the one side and of *monadic logics* on the corresponding side. The main achievements on this respect have been the characterization of regular languages –those recognized by FSMs– in terms of monadic second order logic (MSO) [8] and the definition of an incredible number of subfamilies that are all equivalent between each other and are characterized in terms of monadic first-order logic (MFO) [20].

Such foundational results, however, remained of essentially theoretic interest because the formal correctness problem, though decidable, remains of intractable complexity for MSO and MFO logics

*Work partially supported by project AUTOVAM, funded by Fondazione Cariplo and Regione Lombardia.
†Also with IEIIT, Consiglio Nazionale delle Ricerche.

A. Orlandini, M. Zimmermann (Eds.): 9th Symposium on
Games, Automata, Logics and Formal Verification (GandALF'18)
EPTCS 277, 2018, pp. 161–175, doi:10.4204/EPTCS.277.12

(see e.g., [14]). The state of the art, however, had a dramatic breakthrough with the advent of *temporal logic and model checking* [12]: for now classic logics such as *linear time temporal logic* (LTL), CTL* and others, the model checking problem, though PSPACE complete, has a time complexity bounded by "only" a singly exponential function of formula's length [1].

Not surprisingly, the success of model checking based on FSMs and several extensions thereof, e.g., their timed version [3], generated the wish of extending it to the case of context-free languages (CFLs) to serve much larger application fields such as general purpose programming languages, large web data based on XML, HTML, etc. Whereas the case of the full CFL family is made difficult if not impossible due to the lack of fundamental decidability and closure properties, some early results have been obtained in the context of *structured CFLs*: with this term we mean languages whose typical tree-shaped structure is immediately visible in their sentences; the first instance of such language families are parenthesis languages [19], which correspond to regular tree-languages [21]; among various extensions thereof special attention have received *input-driven* (ID) [7], alias *visibly push-down* languages (VPLs) [4]. Thanks to the fact that they enjoy many of the fundamental closure properties of regular languages, VPLs too have been characterized in terms of a suitable MSO logic [4]; early attempts have also been done to support their model checking by exploiting suitable extensions of temporal logics –whether linear time [1] or branching time [2]. It is also worth mentioning some parallel results on model checking state machines whose nondeterministic computations have a typical tree-shaped structure by means of a variant of alternation-free modal mu-calculus (a branching time logic) [9], whose relationship with pushdown games is discussed in [22], and with both linear and branching time logic in [6].

In this paper we address the same problem in the context of a much larger subfamily of CFL, i.e., *operator precedence languages* (OPLs). OPLs have been invented by R. Floyd to support efficient deterministic parsing of programming languages [13]; subsequently, we showed that they enjoy many of the algebraic properties of structured CFLs [11]; after several decades we resumed the investigation of this family by envisioning their application to various practical state of the art problems, noticeably automatic verification and model checking.[2] We have shown that OPLs strongly generalize VPLs [10] not only in terms of strict set theoretic inclusion but in that they allow to describe typical programming language constructs such as traditional arithmetic expressions whose structure is not immediately "visible" in their sentences, unless one does not take into account the *implicit precedence* of, e.g., multiplicative operators over additive ones. We have built an MSO characterization of OPLs [17] by extending in a non-trivial way the key relation between "matching string positions" introduced in [16] and exploited in [4] for the logical characterization of VPLs. All in all OPLs appear to enjoy many if not all of the pleasant algebraic and logic properties of structured CFLs but widen their application field in a dramatic way. For a more comprehensive description of OPL properties and their relations with other CFL subfamilies see [18].

Here we move one further step in the path toward building model checking algorithms for OPLs –and the wide application field that they can support– of comparable efficiency with other state of the art tools based on the FSM formalism. After resuming the basic background to make the paper self-contained (Section 2), in Section 3 we introduce our *operator precedence temporal logic* (OPTL) which is inspired by temporal logics for nested words, in particular, the logic NWTL[1] but requires many more technicalities due to the lack of the "matching relation" [16] typical of parenthesis languages; we formally define OPTL syntax and semantics and provide examples of its usage and its generality. In Section 4 we show that OPTL defines a larger language family than [1]'s NWTL: every NWTL formula

[1] The parallel field of correctness verification for Turing complete formalisms, instead, ignited many efforts on general purpose "semiautomatic" theorem proving.

[2] We also devised efficient parsers for OPLs by exploiting parallelism [5], but this issue is not the object of the present paper.

can be automatically translated in linear time into an equivalent OPTL formula; strict inclusion follows from the language family inclusion; again, we emphasize that such an inclusion is not just a set theoretical property but shows a much wider application field. Then, in Section 5 we provide the theoretical basis for model checking OP automata (OPAs) against OPTL formulas, i.e., an algorithm which, for any given OPTL formula of length n, builds an equivalent nondeterministic OPA of size 2^n, i.e., the same size of analogous constructions for less powerful automata and less expressive logics. For the sake of brevity in this paper we focus mainly on finite length languages; at the end of the section, however, we provide a few hints showing how our model checking OPAs can be extended to deal with ω-languages. Section 6 concludes and envisages several further research steps.

2 Operator Precedence Languages and Automata

Operator Precedence languages are normally defined through their generating grammars [13]; in this paper, however, we characterize them through their accepting automata [17] which are the natural way to state equivalence properties with logic characterization. We assume some familiarity with classical language theory concepts such as context-free grammar, parsing, shift-reduce algorithm, syntax tree [15].

Let $\Sigma = \{a_1, \ldots, a_n\}$ be an alphabet. The empty string is denoted ε. We use a special symbol # not in Σ to mark the beginning and the end of any string. This is consistent with the operator parsing technique, which requires the look-back and look-ahead of one character to determine the next action [15].

Definition 2.1. *An operator precedence matrix (OPM) M over an alphabet Σ is a partial function $(\Sigma \cup \{\#\})^2 \to \{\lessdot, \doteq, \gtrdot\}$, that with each ordered pair (a,b) associates the OP 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. *Relations $\lessdot, \doteq, \gtrdot$, are respectively named* yields precedence, equal in precedence, *and* takes precedence. *By convention, the initial # can only yield precedence, and other symbols can only take precedence on the ending #. If $M_{a,b} = \circ$, where $\circ \in \{\lessdot, \doteq, \gtrdot\}$, we write $a \circ b$. For $u, v \in \Sigma^+$ we write $u \circ v$ if $u = xa$ and $v = by$ with $a \circ b$.*

Definition 2.2. *An* operator precedence automaton (OPA) *is a tuple $\mathscr{A} = (\Sigma, M, Q, I, F, \delta)$ where:*

- *(Σ, M) is an operator precedence alphabet,*
- *Q is a 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 three disjoint relations:*

$$\delta_{shift} \subseteq Q \times \Sigma \times Q, \quad \delta_{push} \subseteq Q \times \Sigma \times Q, \quad \delta_{pop} \subseteq Q \times Q \times Q.$$

An OPA is deterministic iff I is a singleton, and all three components of δ are –possibly partial– functions: $\delta_{shift} : Q \times \Sigma \to Q$, $\delta_{push} : Q \times \Sigma \to Q$, $\delta_{pop} : Q \times Q \to Q$.

To define the semantics of the automaton, we need some new notations. We use letters $p, q, p_i, q_i, \ldots$ to denote states in Q. We will sometimes use $q_0 \xrightarrow{a} q_1$ for $(q_0, a, q_1) \in \delta_{\text{shift}} \cup \delta_{\text{push}}$, $q_0 \xRightarrow{q_2} q_1$ for $(q_0, q_2, q_1) \in \delta_{\text{pop}}$, 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 let $\Gamma' = \Gamma \cup \{\bot\}$ be the *stack alphabet*; we denote symbols in Γ' as $[a,\ q]$ or $\bot$. We set $smb([a,\ q]) = a$, $smb(\bot) = \#$, and $st([a,\ q]) = q$. Given a stack content $\Pi = \pi_n \ldots \pi_2 \pi_1 \bot$, with $\pi_i \in \Gamma$, $n \geq 0$, we set $smb(\Pi) = smb(\pi_n)$ if $n \geq 1$, $smb(\Pi) = \#$ if $n = 0$.

A *configuration* of an OPA is a triple $c = \langle w,\ q,\ \Pi \rangle$, where $w \in \Sigma^*\#$, $q \in Q$, and $\Pi \in \Gamma^*\bot$.

A *computation* or *run* of the automaton is a finite sequence $c_0 \vdash c_1 \vdash \ldots \vdash c_n$ of *moves* or *transitions* $c_i \vdash c_{i+1}$; there are three kinds of moves, depending on the precedence relation between the symbol on top of the stack and the next symbol to read:

push move: if $smb(\Pi) \lessdot a$ then $\langle ax,\ p,\ \Pi \rangle \vdash \langle x,\ q,\ [a,\ p]\Pi \rangle$, with $(p,a,q) \in \delta_{\text{push}}$;

shift move: if $a \doteq b$ then $\langle bx,\ q,\ [a,\ p]\Pi \rangle \vdash \langle x,\ r,\ [b,\ p]\Pi \rangle$, with $(q,b,r) \in \delta_{\text{shift}}$;

pop move: if $a \gtrdot b$ then $\langle bx,\ q,\ [a,\ p]\Pi \rangle \vdash \langle bx,\ r,\ \Pi \rangle$, with $(q,p,r) \in \delta_{\text{pop}}$.

Observe that shift and pop moves are never performed when the stack contains only $\bot$.

Push and shift moves update the current state of the automaton according to the transition relations δ_{push} and δ_{shift}, respectively: push moves put a new element on top of the stack consisting of the input symbol together with the current state of the automaton, whereas 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 automaton according to δ_{pop} on the basis of the pair of states consisting of the current state of the automaton and the state of the removed stack symbol; pop moves do not consume the input symbol, which is used only to establish the $\gtrdot$ relation, remaining available for the next move.
The automaton accepts the language $L(\mathscr{A}) = \{x \in \Sigma^* \mid \langle x\#,\ q_I,\ \bot \rangle \vdash^* \langle \#,\ q_F,\ \bot \rangle, q_I \in I, q_F \in F\}$.

Definition 2.3. *A* simple chain *is a string $c_0c_1c_2\ldots c_\ell c_{\ell+1}$, written as ${}^{c_0}[c_1c_2\ldots c_\ell]^{c_{\ell+1}}$, such that: c_0, $c_{\ell+1} \in \Sigma \cup \{\#\}$, $c_i \in \Sigma$ for every $i = 1,2,\ldots\ell$ ($\ell \geq 1$), and $c_0 \lessdot c_1 \doteq c_2 \ldots c_{\ell-1} \doteq c_\ell \gtrdot c_{\ell+1}$.*

A composed chain *is a string $c_0s_0c_1s_1c_2\ldots c_\ell s_\ell c_{\ell+1}$, where ${}^{c_0}[c_1c_2\ldots c_\ell]^{c_{\ell+1}}$ is a simple chain, and $s_i \in \Sigma^*$ is the empty string or is such that ${}^{c_i}[s_i]^{c_{i+1}}$ is a chain (simple or composed), for every $i = 0,1,\ldots,\ell$ ($\ell \geq 1$). Such a composed chain will be written as ${}^{c_0}[s_0c_1s_1c_2\ldots c_\ell s_\ell]^{c_{\ell+1}}$.*

The pair made of the first and the last symbols of a chain is called its context.

Definition 2.4. *A finite word w over Σ is* compatible *with an OPM M iff for each pair of letters c,d, consecutive in w, M_{cd} is defined and, for each substring x of $\#w\#$ which is a chain of the form ${}^a[y]^b$, M_{ab} is defined.*

E.g., the word **call handle call call call throw throw throw ret** of Figure 1 is compatible with $M_{\mathbf{call}}$. In the same figure all the resulting chains are reported, e.g. ${}^{\mathbf{call}}[\mathbf{call}]^{\mathbf{throw}}$, ${}^{\mathbf{handle}}[\mathbf{throw}]^{\mathbf{ret}}$ are simple chains, while ${}^{\mathbf{call}}[\mathbf{call}[\mathbf{call}]]^{\mathbf{throw}}$, ${}^{\mathbf{handle}}[[[[\mathbf{call}[\mathbf{call}[\mathbf{call}]]]\mathbf{throw}]\mathbf{throw}]\mathbf{throw}]^{\mathbf{ret}}$ are composed chains.

Definition 2.5. *Let $\mathscr{A}$ be an OPA. We call a* support *for the simple chain ${}^{c_0}[c_1c_2\ldots c_\ell]^{c_{\ell+1}}$ any path in $\mathscr{A}$ of the form $q_0 \xrightarrow{c_1} q_1 \longrightarrow \ldots \longrightarrow q_{\ell-1} \xrightarrow{c_\ell} q_\ell \overset{q_0}{\Longrightarrow} q_{\ell+1}$, where the arrow labeled c_1 corresponds to a push move whereas the remaining ones denote shift moves. 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 ${}^{c_0}[s_0c_1s_1c_2\ldots c_\ell s_\ell]^{c_{\ell+1}}$ *any path in $\mathscr{A}$ of the form*

$$q_0 \overset{s_0}{\rightsquigarrow} q'_0 \xrightarrow{c_1} q_1 \overset{s_1}{\rightsquigarrow} q'_1 \xrightarrow{c_2} \ldots \xrightarrow{c_\ell} q_\ell \overset{s_\ell}{\rightsquigarrow} q'_\ell \overset{q'_0}{\Longrightarrow} q_{\ell+1} \tag{1}$$

where, for every $i = 0,1,\ldots,\ell$: 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 $q'_i = q_i$.

The chains fully determine the structure of the parsing of any automaton over (Σ, M). If the automaton performs the computation $\langle sb, q_i, [a,q_j]\Pi \rangle \vdash^* \langle b, q_k, \Pi \rangle$ then ${}^a[s]^b$ is necessarily a chain over (Σ, M) and there exists a support like (1) with $s = s_0c_1\ldots c_\ell s_\ell$ and $q_{\ell+1} = q_k$. The above computation corresponds to the parsing by the automaton of the string $s_0c_1\ldots c_\ell s_\ell$ within the context a,b. Such context contains all information needed to build the subtree whose frontier is that string. This is a distinguishing feature of OP languages: we call it the *locality principle*.

	call	**ret**	**handle**	**throw**
call	$\lessdot$	$\doteq$	$\lessdot$	$\gtrdot$
ret	$\gtrdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
handle	$\lessdot$	$\gtrdot$	$\lessdot$	$\lessdot$
throw	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$

#[**call**[**handle**[[[[**call**[**call**[**call**]]]**throw**]**throw**]**throw**]]**ret**]#

Figure 1: OPM $M_{\textbf{call}}$ and an example string with all the chains evidenced by brackets.

Definition 2.6. *Given* (Σ, M)*, let us consider the OPA* $\mathscr{A}(\Sigma, M) = \langle \Sigma, M, \{q\}, \{q\}, \{q\}, \delta_{max} \rangle$ *where* $\delta_{max}(q,q) = q$*, and* $\delta_{max}(q,c) = q$*,* $\forall c \in \Sigma$*. We call* $\mathscr{A}(\Sigma, M)$ *the* OP Max-Automaton *over* Σ, M*.*

For a max-automaton $\mathscr{A}(\Sigma, M)$ each chain has a support; since there is a chain $^{\#}[s]^{\#}$ for any string s compatible with M, a string is accepted by $\mathscr{A}(\Sigma, M)$ iff it is compatible with M. Also, whenever M is complete, each string is compatible with M, hence accepted by the max-automaton; thus, when M is complete the max-automaton defines the universal language Σ^* by assigning to any string the (unique) structure compatible with the OPM. Considering $M_{\textbf{call}}$ of Figure 1, if we take e.g. the string **ret call handle**, it is accepted by the max-automaton and its structure is #[[**ret**]**call**[**handle**]]#.

In conclusion, given an OP alphabet, the OPM M assigns a structure to any compatible string in Σ^*; unlike parentheses languages 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 strings compatible with M. In some sense this property is yet another variation of the fundamental Chomsky-Shützenberger theorem. All above definitions are extended to the case of infinite strings in the traditional way [17]; for this reason and for the sake of brevity we will deal with the ω-case only in Section 5.1 where the application of novel techniques is necessary. For a more complete description of the OPL family and of its relations with other CFLs we refer the reader to [18].

3 Operator Precedence Temporal Logic

Languages recognized by different OPAs on a given OP alphabet form a Boolean algebra [11], i.e. they are closed under union, intersection and complement: these properties allow us to define OPTL (*Operator Precedence Temporal Logic*). Given an OP alphabet, each well-formed OPTL formula characterizes a subset of the universal language based on that alphabet. Due to the closure properties above, for each OPTL formula it is possible to identify an OPA that recognizes the same language denoted by it, opening the way for model checking of OPTL.

Next, we present the syntax of OPTL explaining its meaning by means of simple examples, then formally define its semantics. We explore the relationships between the OPTL operators in Section 3.2.

3.1 Syntax and Semantics

OPTL is a propositional temporal logic: the truth of formulas depends on the atomic propositions holding in each word position. Let AP be the finite set of atomic propositions: the semantics of OPTL relies on an OP alphabet based on $\mathscr{P}(AP)$, so terminal characters are subsets of AP, and word structure is given by an OPM defined on $\mathscr{P}(AP)$. Since defining an OPM on a power set is often uselessly cumbersome, in this paper we define it on a set $\Sigma \subseteq AP$, whose elements are called "structural labels", and typeset in bold. The corresponding OPM on $\mathscr{P}(AP)$ can be derived from the structural label contained in each subset of AP, leaving the matrix undefined for subsets not containing exactly one element of Σ. For example,

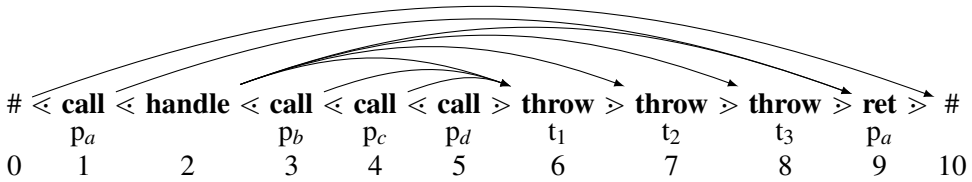


Figure 2: An example of execution trace, according to $M_{\textbf{call}}$ (Figure 1). Chains are highlighted by arrows joining their context; structural labels are typeset in bold, while other atomic propositions are shown below them. First, procedure p_a is called (pos. 1), and it installs an exception handler in pos. 2. Then, three nested procedures are called, and the innermost one (p_d) throws a sequence of exceptions, which are all caught by the handler. Finally, p_a returns, uninstalling the handler.

in the word of Figure 2, pos. 1 is labeled with the set $\{\textbf{call}, \mathrm{p}_a\}$, and pos. 3 with $\{\textbf{call}, \mathrm{p}_b\}$: they both contain **call**, so they are in the $\lessdot$ relation according to matrix $M_{\textbf{call}}$ of Figure 1.

Syntax. The syntax of OPTL is based on the following grammar, where a denotes any symbol in *AP*:

$$\varphi := \mathrm{a} \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \bigcirc\varphi \mid \bigcirc_\chi\varphi \mid \ominus\varphi \mid \ominus_\chi\varphi \mid$$
$$(\varphi \,\mathscr{U}\, \varphi) \mid (\varphi \,\mathscr{U}^{\boxdot}\, \varphi) \mid (\varphi \,\mathscr{S}\, \varphi) \mid (\varphi \,\mathscr{S}^{\boxdot}\, \varphi) \mid (\varphi \,\mathscr{U}^{\circledcirc}\, \varphi) \mid (\varphi \,\mathscr{S}^{\circledcirc}\, \varphi)$$

We informally show the meaning of OPTL operators by referring to the word of Figure 2, w.r.t the OPM of Fig. 1. The $\bigcirc$ and $\ominus$ symbols denote the next and back operators from LTL, while the undecorated $\mathscr{U}$ and $\mathscr{S}$ operators are LTL until and since. The $\bigcirc_\chi$ and $\ominus_\chi$ operators, which we call *matching next* and *matching back*, express properties on string positions in the chain relation (which will be formally defined later on) with the current one. For example, formula $\bigcirc_\chi$**throw** is true in positions containing a **call** to a procedure that is terminated by an exception thrown by an inner procedure, such as 3 and 4 of Fig. 2, because pos. 3 forms a chain with pos. 6, in which **throw** holds, and so on. Formula $\ominus_\chi$**handle** is true in handled **throw** positions, such as 6, 7 and 8, because e.g. pos. 2 forms a chain with 6, and **handle** holds in 2. The $\mathscr{U}^{\boxdot}$ and $\mathscr{S}^{\boxdot}$ operators, called *operator precedence summary until and since*, are inspired to the homonymous $\mathscr{U}^{\sigma}$ and $\mathscr{S}^{\sigma}$ operators from [1], and are path operators that can "jump" over chain bodies; the symbol $\boxdot$ is a placeholder for one or more precedence relations allowed in the path (e.g. $\mathscr{U}^{\lessdot\doteq}$ or $\mathscr{U}^{\gtrdot\lessdot}$ and so on). Formula (**call** $\vee$ **throw**) $\mathscr{U}^{\gtrdot}$ **ret** is true in pos. 3 because there is a path that jumps over the chain between 3 and 6, and goes on with positions 7, 8 and 9, which are in the $\gtrdot$ relation: pos. 3 and from 6 to 8 satisfy **call** $\vee$ **throw**, while pos. 9 satisfies **ret**. In $\mathscr{U}^{\circledcirc}$ and $\mathscr{S}^{\circledcirc}$, $\circledcirc$ is a placeholder for $\uparrow$ or $\downarrow$; these peculiar path operators are called *hierarchical until and since*, and they express properties about the multiple positions in the chain relation with the current one: their associated paths can dive up and down between such positions. For example, **throw** $\mathscr{U}^{\uparrow}\,\mathrm{t}_3$ and **throw** $\mathscr{S}^{\downarrow}\,\mathrm{t}_1$ hold in pos. 2, because there is path 6-7-8 made of ending positions of chains starting in 2, such that **throw** holds until t_3 holds (or **throw** has held since t_1 held). Formulas **call** $\mathscr{U}^{\downarrow}\,\mathrm{p}_c$ and **call** $\mathscr{S}^{\uparrow}\,\mathrm{p}_b$ hold in pos. 6, because of path 3-4, made of positions where a chain ending in 6 starts, and whose labels satisfy the appropriate until and since conditions.

Many relevant properties can be expressed in OPTL: formula $\Box[\textbf{handle} \implies \bigcirc_\chi\textbf{ret}]$, where $\Box\psi$ is a shortcut for $\neg(\top\,\mathscr{U}\,\neg\psi)$, holds if all exception handlers are properly uninstalled by a return statement. Formula $\Box[\textbf{throw} \implies \neg(\top\,\mathscr{S}^{\uparrow}\,\mathrm{p}_b)]$ is false if procedure p_b is terminated by an exception; and formula $\neg(\top\,\mathscr{U}^{\uparrow}\,(\textbf{throw} \wedge \top\,\mathscr{U}^{\downarrow}\,\textbf{call}))$ is true in **handle**s catching only throw statements not interrupting any procedure. These properties may not be expressible in NWTL, because its nesting relation is one-to-one,

and fails to model situations in which a single entity is in relation with multiple other entities.

Semantics. The OPTL semantics is based on the OP word structure $\langle U, M_{\mathscr{P}(AP)}, P\rangle$ where

- $U = \{0,1,\ldots,n,n+1\}$, with $n \in \mathbb{N}$ is a set of word positions;
- $M_{\mathscr{P}(AP)}$ is an operator precedence matrix on $\mathscr{P}(AP)$;
- $P\colon U \to \mathscr{P}(AP)$ is a function associating each word position in U with the set of atomic propositions that hold in that position, with $P(0) = P(n+1) = \{\#\}$.

The word structure is given by the OPM $M_{\mathscr{P}(AP)}$, which associates a precedence relation to each pair of positions, based on the subset of AP associated to them by P. For OPTL to be able to denote the universal language $\mathscr{P}(AP)^*$, $M_{\mathscr{P}(AP)}$ must be complete. In the following we will denote subsets of AP by lowercase letters in italic, such as $a \in \mathscr{P}(AP)$. For any $i, j \in U$ we write, $i \lessdot j$ if $a = P(i)$, $b = P(j)$ and the relation $a \lessdot b$ is in $M_{\mathscr{P}(AP)}$.

The semantics of OPTL deeply relies on the concept of *chain*, presented in Section 2. We define the chain relation $\chi \subseteq U \times U$ so that $\chi(i,j)$ holds between two positions $i < j$ if i and j form the context of a chain. In case of composed chains, this relation is not one-to-one: there may be positions where multiple chains start or end. Therefore, we additionally define two one-to-one relations, helpful in identifying the largest chain starting or ending in a word position. More formally, the *maximal forward* chain relation is defined so that $\overrightarrow{\chi}(i,j) \iff j = \max\{k \in U \mid \chi(i,k)\}$ for any $i, j \in U$, and the *maximal backward* chain is defined as $\overleftarrow{\chi}(i,j) \iff i = \min\{k \in U \mid \chi(k,j)\}$. The max. forward (resp. backward) chain relation can be undefined for a pair of positions if either they are the context of no chain, or if they are the context of a chain which is not forward- (resp. backward-) maximal.

Let w be an OP word, and $a \in AP$. Then, for any position $i \in U$ of w, we have $(w,i) \models a$ if $a \in P(i)$. Operators such as $\wedge$ and $\neg$ have the usual semantics from propositional logic, while $\bigcirc$ and $\ominus$ have the same semantics as in LTL (i.e. $(w,i) \models \bigcirc\varphi$ iff $(w,i+1) \models \varphi$, and similarly for $\ominus$).

The $\bigcirc_\chi$ and $\ominus_\chi$ operators express properties regarding the positions that form a maximal chain that starts (resp. ends) in the current one: $(w,i) \models \bigcirc_\chi\varphi$ iff there exists a position $j \in U$ such that $\overrightarrow{\chi}(i,j)$ and $(w,j) \models \varphi$; symmetrically, $(w,i) \models \ominus_\chi\varphi$ iff there exists a position $j \in U$ such that $\overleftarrow{\chi}(j,i)$ and $(w,j) \models \varphi$. In Figure 2, $(w,3) \models \bigcirc_\chi$**throw** because $\overrightarrow{\chi}(3,6)$ and $(w,6) \models$ **throw**; $(w,6) \models \ominus_\chi$**handle** holds because $\overleftarrow{\chi}(2,6)$ and $(w,2) \models$ **handle**, but $(w,6) \not\models \ominus_\chi \mathrm{p}_b$ because chain $\chi(3,6)$ is not backward-maximal (although it is forward-maximal).

A *path* of length $n \in \mathbb{N}$ between $i, j \in U$ is a sequence of positions $i_1 < i_2 < \cdots < i_n$, with $i \leq i_1$ and $i_n \leq j$. The *until* operator on a set of paths Π is defined as follows: for any word w and position $i \in U$, and for any two OPTL formulas φ and ψ, $(w,i) \models \varphi\,\mathscr{U}^{\Pi}\,\psi$ 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 Π such that $(w,i_k) \models \varphi$ for any $1 \leq k < n$, and $(w,i_n) \models \psi$. The *since* operator is defined symmetrically. Note that a path from i to j does not necessarily start in i and end in j, but it may do in positions between them. However, this will only happen with hierarchical paths. We define the different kinds of until/since operators by associating them with suitable set of paths.

The *linear until* ($\varphi\,\mathscr{U}\,\psi$) and *since* ($\varphi\,\mathscr{S}\,\psi$) operators, based on linear paths, have the same semantics as in LTL. A *linear path* starting in position $i \in U$ is such that $i_1 = i$ and for any $1 \leq k < n$ $i_{k+1} = i_k + 1$.

The *OP-summary until* operator exploits the $\overrightarrow{\chi}$ relation to express properties on paths that skip chain bodies, also keeping precedence relations between consecutive word positions into account.

Definition 3.1. *Given a set $O \subseteq \{\lessdot, \doteq, \gtrdot\}$, the $\mathscr{U}^O$ operator is based on the class of* forward OP-summary *paths. A path of this class between i and $j \in U$ is a sequence of positions $i = i_1 < i_2 < \cdots <$*

$i_n = j$ *such that, for any* $1 \leq k < n$,

$$i_{k+1} = \begin{cases} h & \text{if } \overrightarrow{\chi}(i_k, h) \text{ and } h \leq j; \\ i_k + 1 & \text{if } i_k \odot i_k + 1 \text{ with } \odot \in O, \text{ otherwise.} \end{cases}$$

There exists at most one forward OP-summary path between any two positions. For example, in Figure 2, if we take $O = \{\gtrdot\}$ as in $(\mathbf{call} \vee \mathbf{throw})\, \mathscr{U}^{\gtrdot}\, \mathbf{ret}$, the path between 3 and 9 is made of pos. 3-6-7-8-9, because $\overrightarrow{\chi}(3,6)$ and the body of this chain is skipped, and $6 \gtrdot 7$, $7 \gtrdot 8$ and $8 \gtrdot 9$. If we took e.g. $O = \{\doteq, \lessdot\}$, there would be no such path, because consecutive positions in the $\gtrdot$ relation are not considered. With $O = \{\gtrdot, \lessdot\}$, the path between 2 and 6 does not skip the body of chain $\chi(2,6)$, because it is not forward-maximal: it is the linear path 2-3-4-5-6. The *OP-summary since* operator is based on *backward OP-summary* paths, which are symmetric to their until counterparts, relying on the $\overleftarrow{\chi}$ relation instead of $\overrightarrow{\chi}$. For example, $(\mathbf{throw} \vee \mathbf{handle})\, \mathscr{S}^{\lessdot}\, \mathbf{call}$ holds in pos. 8 because of path 1-2-8, that skips the body of chain $\overleftarrow{\chi}(2,8)$ and satisfies $\mathbf{throw} \vee \mathbf{handle}$ in 2 and 8, and **call** in 1. Again, bodies of chains that are not backward-maximal cannot be skipped.

While summary operators are only aware of the maximal chain relations, hierarchical operators can express properties discriminating between all other chains. The *hierarchical yield-precedence* until and since operators, denoted as $\mathscr{U}^{\uparrow}$ and $\mathscr{S}^{\downarrow}$ respectively, are based on paths made of the ending positions of non-maximal chains starting in the current position $i \in U$. One of such paths is a sequence of word positions $i_1 < i_2 < \cdots < i_n$, with $i < i_1$, such that for any $1 \leq k \leq n$ we have $i \lessdot i_k$ and $\chi(i, i_k)$, and, additionally, there is no i'_k that satisfies these two properties and $i_{k-1} < i'_k < i_k$. Moreover, for the until operator i_1 must be the leftmost position enjoying the above properties (i.e. there is no i'_1 s.t. $i < i'_1 < i_1$ enjoying them), and for the since operator i_n must be the rightmost. The latter is a since operator despite being a future modality. We chose this naming because in formulas such as $\varphi\, \mathscr{S}^{\downarrow}\, \psi$, ψ must hold at the beginning of the path, while φ must hold in subsequent positions, which is the typical behavior of since operators. Note that these paths only contain forward non-maximal chain ends, which are in the $\lessdot$ relation with i. For example, in pos. 2 $\mathbf{throw}\, \mathscr{U}^{\uparrow}\, \mathrm{t}_3$ holds because of path 6-7-8, since **throw** holds in 6-7 and t_3 in 8; instead, the path made only of pos. 6 and the one made of pos. 6-7, in which t_3 does not hold, do not satisfy it. Similarly, $\mathbf{throw}\, \mathscr{S}^{\downarrow}\, \mathrm{t}_1$ is satisfied by path 6-7-8, but not by the path made of pos. 8 and the one made of pos. 7-8, in which t_1 does not hold. Position 9 is not included in these paths, because it does not satisfy the condition $2 \lessdot 9$ (indeed, $2 \gtrdot 9$).

Conversely, *hierarchical take-precedence* until and since operators ($\mathscr{U}^{\downarrow}$ and $\mathscr{S}^{\uparrow}$) consider non-maximal chains ending in the current position $j \in U$. The paths they are based on are sequences of word positions $i_1 < i_2 < \cdots < i_n$, with $i_n < j$, such that for any $1 \leq k \leq n$ we have $i_k \gtrdot j$ and $\chi(i_k, j)$, and, additionally, there exists no position i'_k that satisfies these two properties and $i_k < i'_k < i_{k+1}$. For the until operator, i_1 must be the leftmost position enjoying these properties, and for the since operator i_n must be the rightmost (i.e. there is no i'_n, $i_n < i'_n < j$, that satisfies them). Note that $\mathscr{U}^{\downarrow}$ is an until operator despite being a past modality: again, the reason is that $\varphi\, \mathscr{U}^{\downarrow}\, \psi$ enforces ψ at the end of the path, and φ in previous positions, making it more similar to an until operator. In pos. 6, $\mathbf{call}\, \mathscr{U}^{\downarrow}\, \mathrm{p}_c$ is satisfied by path 3-4 and not by the one made only of pos. 3, because p_c does not hold in 3, but it does in 4, and **call** holds in 3. Formula $\mathbf{call}\, \mathscr{S}^{\uparrow}\, \mathrm{p}_b$ is satisfied by path 3-4, and not by the one made only of 4, because p_b holds in 3 and **call** in 4.

3.2 Equivalence between Operators

In this section, we show that hierarchical operators do not contribute to the expressiveness of OPTL, because OP-summary until operators can be used in place of them, at the expense of an exponential

blowup in formula length. First, we define two auxiliary formulas:

$$\sigma_a \equiv \Big(\bigwedge_{\ell\in a}\ell\Big)\wedge\Big(\bigwedge_{\ell\in(AP\setminus a)}\neg\ell\Big),\qquad \xi_{a\lessdot}\equiv\bigvee_{b\in\mathscr{P}(AP)|a\lessdot b}\sigma_b.$$

For any $a\in\mathscr{P}(AP)$, σ_a holds only in positions labeled with a. $\xi_{a\lessdot}$ holds only in positions labeled with a set $b\in\mathscr{P}(AP)$ such that $a\lessdot b$, and allows to express OP relations in OPTL formulas. Formula $\xi_{\gtrdot a}$ is defined similarly. The yield-precedence hierarchical until operator $\mathscr{U}^{\uparrow}$ can be translated as follows:

$$\varphi\,\mathscr{U}^{\uparrow}\psi\equiv\bigvee_{a\in\mathscr{P}(AP)}\Big[\sigma_a\wedge\bigcirc\Big(\big(\ominus_\chi\sigma_a\implies\varphi\big)\,\mathscr{U}^{\gtrdot\doteq}\big(\ominus_\chi\sigma_a\wedge\xi_{a\lessdot}\wedge\psi\big)\Big)\Big].$$

Let us say the formula is evaluated in position $i\in U$, labeled with $a\in\mathscr{P}(AP)$. The $\mathscr{U}^{\uparrow}$ operator considers paths made of positions k such that $\chi(i,k)$ and $i\lessdot k$, i.e. ends of non-maximal chains starting in i. φ must hold in all positions of a path except the last one, j, in which ψ must hold. This translation works by letting an OP-summary path run from $i+1$ to a position $h=j$. Formula $\ominus_\chi\sigma_a$ makes sure h is the end of a chain starting in a position labeled with a, which must be i. Moreover, $\xi_{a\lessdot}$ implies $i\lessdot h$, and ψ must hold in h. Therefore, we have $h=j$. In all positions $i<k<j$ such that $\chi(i,k)$, φ must hold: this is enforced similarly by subformula $\ominus_\chi\sigma_a\implies\varphi$. The fact that $\chi(i,j)$ implies $i\lessdot k$ for all positions k above, so there is no need to put $\xi_{a\lessdot}$ in the left side of the until operator. Since the OP relations between two positions depend on the subsets of AP labeling them, the disjunction of separate instances of the formula for each $a\in\mathscr{P}(AP)$ are needed.

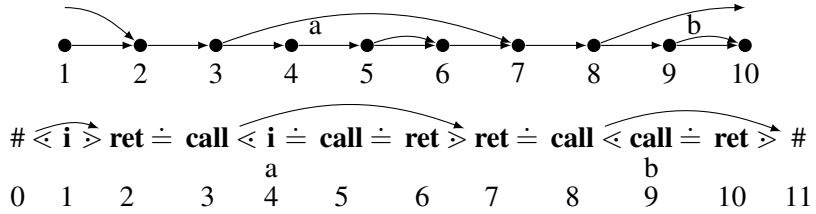
Formulas translating other hierarchical operators are analogous. Notice that the size of the translation above is exponential in the length of the initial formula. In fact, let $n=|AP|$: $|\sigma_a|=\Theta(n)$ and $|\xi_{a\lessdot}|=O(n2^n)$. If β is the translation of $\alpha=\varphi\,\mathscr{U}^{\uparrow}\psi$, then in the worst-case scenario we have $|\beta|=O((2^n)^{|\alpha|}|\xi_{a\lessdot}|)=O(n2^{2n|\alpha|})$.

The above result allows us to state that the set of operators $\{\neg,\wedge,\bigcirc,\ominus,\bigcirc_\chi,\ominus_\chi,\mathscr{U},\mathscr{S},\mathscr{U}^{\boxdot},\mathscr{S}^{\boxdot}\}$ is adequate for OPTL. We conjecture that a result similar to the above could be obtained for the linear until and since operators as well, allowing us to omit them from this set.

4 Relationship with Nested Words

We now explore the relationship between OPTL and the NWTL logic presented in [1]. NWTL is a temporal logic based on the VPL family, which is strictly contained in OPL. In [10, 17, 18] the relations between the two families are discussed in depth both from a mathematical and an application point of view: building OPTL formulas describing OPLs that are not VPLs is a trivial job. This proves that there exist languages not expressible in NWTL that can be expressed in OPTL. To prove that OPTL is more expressive than NWTL, we first show a way to translate a nested word into an "almost isomorphic" OPTL structure; then, we give a translation schema for NWTL formulas into equivalent OPTL ones.

Recall that a nested word is a structure $NW=\langle U,(P_{\mathrm{a}})_{\mathrm{a}\in\Lambda},<,\mu,\texttt{call},\texttt{ret}\rangle$, where Λ is the set of atomic propositions; U is a set of word positions such that $U=\{1,\ldots,n\}$ if NW is finite, and $U=\mathbb{N}$ if it is a nested ω-word; '$<$' is the ordering of $\mathbb{N}$; P_{a} is the set of positions labeled with a. μ is a binary relation and `call` and `ret` are unary relations such that $\mu(i,j)$ implies $\texttt{call}(i)$ and $\texttt{ret}(j)$; if $\mu(i,j)$ and $\mu(i,j')$ hold then $j=j'$, while if $\mu(i,j)$ and $\mu(i',j)$ then $i=i'$; and if $i\leq j$ and $\texttt{call}(i)$ and $\texttt{ret}(j)$ then there exists a position k such that $i\leq k\leq j$, and either $\mu(i,k)$ or $\mu(k,j)$. If $\mu(i,j)$ then i is the matching



	call	ret	i
call	⋖	≐	⋖
ret	≐	⋗	≐
i	≐	⋗	≐

Figure 3: The top-left figure is the representation of a nested word example, and its translation into an OPTL structure is shown below, w.r.t. the OPM M^{NW} on the right.

call of j, which is the matching return of i. If $\texttt{call}(i)$ (resp. $\texttt{ret}(j)$) but for no $j \in U$ (resp. $i \in U$) we have $\mu(i,j)$, then i (resp. j) is a *pending* call (resp. return).

The syntax of NWTL is given by $\varphi := \top \mid \mathrm{a} \mid \texttt{call} \mid \texttt{ret} \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \bigcirc_\mu\varphi \mid \ominus\varphi \mid \ominus_\mu\varphi \mid \varphi\,\mathscr{U}^\sigma\,\varphi \mid \varphi\,\mathscr{S}^\sigma\,\varphi$, with $\mathrm{a} \in \Lambda$. The semantics of propositional and LTL-like operators is the usual one. For any $i \in U$, $(NW,i) \models \texttt{call}$ iff $\texttt{call}(i)$ and similarly for $\texttt{ret}$; $(NW,i) \models \bigcirc_\mu\varphi$ iff there exists $j \in U$ such that $\mu(i,j)$ and $(NW,j) \models \varphi$, while the meaning of $\ominus_\mu\varphi$ is analogous, but it refers to the past. The until and since operators $\mathscr{U}^\sigma$ and $\mathscr{S}^\sigma$ are based on summary paths. A summary path between $i,j \in U$, $i < j$, is a sequence of positions $i = i_0 < i_1 < \cdots < i_n = j$ such that for any $0 \le k < n$ we have either $i_{k+1} = h$ if $\mu(i_k,h)$ and $h \le j$, or $i_{k+1} = i_k + 1$ otherwise.

Given any nested word NW as defined above, it is possible to build an equivalent algebraic structure for OPTL as $OW = \langle U', M^{NW}, P' \rangle$. Given $U = \{1,\ldots,n\}$, we have $U' = U \cup \{0, n+1\}$. The set of propositional letters is $AP = \Lambda \cup \Sigma$ with $\Sigma = \{\mathbf{call}, \mathbf{ret}, \mathbf{i}\}$. For any $i \in U$ we define $P'(i) = \{\mathrm{a} \in \Lambda \mid i \in P_\mathrm{a}\} \cup \sigma(i)$, where $\sigma(i) = \{\mathbf{call}\}$ iff $\texttt{call}(i)$, $\sigma(i) = \{\mathbf{ret}\}$ iff $\texttt{ret}(i)$, and $\sigma(i) = \{\mathbf{i}\}$ otherwise. Finally, the OPM M^{NW} is shown in Figure 3.

An example nested word is shown in Figure 3, along with its translation into an OPTL word. In the translation, all the call positions form the context of a chain with the matched return, except for consecutive positions $i, i+1 \in U$ such that i is a call and $i+1$ a return. Therefore, we are able to use the chain relation to translate the matching relation of nested words, except for consecutive call/return positions, which have to be considered separately. Also, unmatched returns and calls form a chain with the first and last # positions. This is formalized by the following properties, whose proofs are omitted:

1. For any two distinct positions $i, j \in U$, $i < j$, if $\chi(i,j)$ holds then $\mathbf{call} \in P'(i)$ and $\mathbf{ret} \in P'(j)$.
2. For any $i, j, j' \in U$ if $\chi(i,j)$ and $\chi(i,j')$ then $j = j'$, and for any $i, i', j \in U$ if $\chi(i,j)$ and $\chi(i',j)$ then $i = i'$.
3. For any $i, j \in U$ such that $j > i+1$, we have $\mu(i,j)$ iff $\chi(i,j)$.
4. Given any two positions $i, j \in U$, $i \le j$, the summary path between i and j in NW coincides with the OP-summary path between the same positions based on OP relations $O = \{\lessdot, \doteq, \gtrdot\}$ in OW.

After establishing a certain degree of isomorphism between nested words and their OPTL translations, we can give a translation schema from NWTL to OPTL formulas.

Theorem 4.1 (NWTL $\subseteq$ OPTL)**.** *Given an NWTL formula φ, it is possible to translate it to an OPTL formula φ' of length linear in $|\varphi|$ such that, for any nested word w and position i, if w is translated into an OP word w' as described at the beginning of Section 4, then $(w,i) \models \varphi$ iff $(w',i) \models \varphi'$, with $i \in U$.*

Proof. Let w' be an OP word built from w as described above. For any NWTL formula φ we denote as $\varphi' = \alpha(\varphi)$ the OPTL formula that satisfies $(w,i) \models \varphi$ iff $(w',i) \models \varphi'$. The translation function α is defined for non-trivial arguments as follows:

• $\alpha(\bigcirc_\mu \varphi) = (\mathbf{call}\, \mathscr{U}^{\doteq}\, (\mathbf{ret} \wedge \alpha(\varphi))) \wedge \neg\mathbf{ret}$, and $\alpha(\ominus_\mu \varphi) = (\mathbf{ret}\, \mathscr{S}^{\doteq}\, (\mathbf{call} \wedge \alpha(\varphi))) \wedge \neg\mathbf{call}$. [3]

Let $\gamma \equiv \mathbf{call}\, \mathscr{U}^{\doteq}\, (\mathbf{ret} \wedge \alpha(\varphi))$. In NWTL we have $(w,i) \models \bigcirc_\mu \varphi$ iff there exists $j \in U$ such that $\mu(i,j)$ and $(w,j) \models \varphi$. Because of Prop. 3, if such a position j exists and $j > i+1$ then also $\chi(i,j)$ holds and, because of Prop. 2, we have $\overrightarrow{\chi}(i,j)$. Consider the path only made of i and j: it is an OP summary path, because it falls in the first case of the definition. Since by construction $\mathbf{call} \in P'(i)$ and $\mathbf{ret} \in P'(j)$, if $\alpha(\varphi)$ holds in j, then γ is satisfied, and this is the only path in which γ is true. Paths terminating in a position strictly between i and j are forbidden by allowing only the $\doteq$ relation; also, all paths surpassing j must include it, but $\mathbf{call} \notin P'(j)$ falsifies γ. If i and j are such that $\mu(i,j)$ but $j = i+1$, we have $\mathbf{call} \in P'(i)$ and $\mathbf{ret} \in P'(j)$, so $i \doteq j$, and the path made of i and j is valid. Finally, if $(w,i) \not\models \bigcirc_\mu \varphi$ because position i is not a matched call, then $\mathbf{i} \in P'(i)$ and γ is false because $\mathbf{call}, \mathbf{ret} \notin P'(i)$. If i is an unmatched call, then $\chi(i,n+1)$, but $\mathbf{ret} \notin P'(n+1)$, which falsifies γ. If $\mathtt{ret}(i)$, then $\neg\mathbf{ret}$ is false. The translation for $\ominus_\mu$ is similar.

• $\alpha(\varphi\, \mathscr{U}^{\sigma}\, \psi) = \alpha(\varphi)\, \mathscr{U}^{\gtrdot \doteq \lessdot}\, \alpha(\psi)$ and $\alpha(\varphi\, \mathscr{S}^{\sigma}\, \psi) = \alpha(\varphi)\, \mathscr{S}^{\gtrdot \doteq \lessdot}\, \alpha(\psi)$: from Prop. 4 we know that the set of summary paths starting from position i in w corresponds to the set of OP summary paths starting from i in w', which implies the OP summary operators coincide with their NWTL counterparts.

By induction on the syntactic structure of φ, we can conclude NWTL $\subseteq$ OPTL. □

For example, take formula $\varphi = (\neg \mathrm{a})\, \mathscr{U}^{\sigma}\, \mathrm{b}$: the nested word of Fig. 3 satisfies φ because of summary path 1-2-3-7-8-9. φ is translated into $\varphi' = (\neg \mathrm{a})\, \mathscr{U}^{\gtrdot \doteq \lessdot}\, \mathrm{b}$, which is satisfied by the OPTL structure of Fig. 3, where the OP summary path covering the same positions above witnesses its truth.

5 Model Checking Operator Precedence Languages

Given a set of atomic propositions AP, an OPM $M_{\mathscr{P}(AP)}$, and an OPTL formula φ containing atomic propositions in AP, we explain how to build an automaton $\mathscr{A}_\varphi$ accepting the finite strings on $\mathscr{P}(AP)$ satisfying φ, thus compatible with $M_{\mathscr{P}(AP)}$. We later sketch the way to extend this definition to infinite words. Let $\mathscr{A}_\varphi = \langle \mathscr{P}(AP), M_{\mathscr{P}(AP)}, \mathrm{Atoms}(\varphi), I, F, \delta \rangle$ be an OPA, with $M_{\mathscr{P}(AP)}$ being an OPM, and $\mathrm{Atoms}(\varphi)$ being the set of states. We first describe how the OPA is built for LTL-like operators, which is done with the classic procedure by [23], and then treat other operators separately. Initially, the *closure* $\mathrm{Cl}(\varphi)$ of a formula φ is defined as the smallest set such that $\varphi \in \mathrm{Cl}(\varphi)$, $AP \subseteq \mathrm{Cl}(\varphi)$, if $\psi \in \mathrm{Cl}(\varphi)$ and $\psi \neq \neg\theta$ for any OPTL formula θ, then $\neg\psi \in \mathrm{Cl}(\varphi)$; if any of $\neg\psi$, $\bigcirc\psi$ or $\ominus\psi$ is in $\mathrm{Cl}(\varphi)$, then $\psi \in \mathrm{Cl}(\varphi)$; if any of $\psi \wedge \theta$ or $\psi \vee \theta$ is in $\mathrm{Cl}(\varphi)$, then $\psi \in \mathrm{Cl}(\varphi)$ and $\theta \in \mathrm{Cl}(\varphi)$. We will further enrich $\mathrm{Cl}(\varphi)$ when dealing with non-LTL operators. The set $\mathrm{Atoms}(\varphi)$ contains all consistent sets $\Phi \subseteq \mathrm{Cl}(\varphi)$, i.e., such that for every $\psi \in \mathrm{Cl}(\varphi)$, $\psi \in \Phi$ iff $\neg\psi \notin \Phi$; if $\psi \wedge \theta \in \Phi$, then $\psi \in \Phi$ and $\theta \in \Phi$; if $\psi \vee \theta \in \Phi$, then $\psi \in \Phi$ or $\theta \in \Phi$. The set of initial states I consists only of atoms containing φ, but with no $\ominus$ formula. The set of final states F contains all atoms $\Phi \in \mathrm{Atoms}(\varphi)$ such that for any $\bigcirc\psi \in \mathrm{Cl}(\varphi)$, $\bigcirc\psi \notin \Phi$, so no word terminating with unsatisfied temporal requirements can be accepted, and $AP \cap \Phi = \{\#\}$, so the last position of each word must contain the terminal symbol only. Satisfaction of temporal constraints is achieved by the transition relation δ. For any $\Phi, \Theta \in \mathrm{Atoms}(\varphi)$ and $a \in \mathscr{P}(AP)$, $(\Phi, a, \Theta) \in \delta_{\text{push}} \cap \delta_{\text{shift}}$ iff for any $\mathrm{p} \in AP$, $\mathrm{p} \in \Phi$ iff $\mathrm{p} \in a$; $\bigcirc\psi \in \Phi$ iff $\psi \in \Theta$; $\ominus\psi \in \Theta$ iff $\psi \in \Phi$, for any formula ψ. Since only push and shift transitions read terminal symbols, they fulfill next and back requirements. Pop transitions are important for checking operators depending on the chain relation, but for the moment we prevent them from removing sub-formulas introduced in a previous state by other transitions: for any

[3] We chose these translations instead of the more straightforward $\alpha(\bigcirc_\mu \varphi) = \bigcirc_\chi \alpha(\varphi) \vee (\mathbf{call} \wedge \bigcirc(\mathbf{ret} \wedge \alpha(\varphi))$ (and an analogous one for $\ominus_\mu \varphi$) because the latter causes an exponential blowup in formula length.

	a	**b**	**c**
a	$\gtrdot$	$\lessdot$	$\doteq$
b	$\gtrdot$	$\gtrdot$	$\gtrdot$
c	$\gtrdot$	$\gtrdot$	$\lessdot$

step	input	state	stack
1	$\mathbf{a} \lessdot \mathbf{b} \gtrdot \mathbf{b} \gtrdot \mathbf{c} \gtrdot \#$	$\{\mathbf{a}, \bigcirc_\chi \mathbf{c}\}$	$\bot$
2	$\mathbf{b} \gtrdot \mathbf{b} \gtrdot \mathbf{c} \gtrdot \#$	$\{\mathbf{b}, \bigcirc^{\mathrm{s}}_\chi \mathbf{c}\}$	$[\mathbf{a}, \{\mathbf{a}, \bigcirc_\chi \mathbf{c}\}]\bot$
3	$\mathbf{b} \gtrdot \mathbf{c} \gtrdot \#$	$\{\mathbf{b}\}$	$[\mathbf{b}, \{\mathbf{b}, \bigcirc^{\mathrm{s}}_\chi \mathbf{c}\}][\mathbf{a}, \{\mathbf{a}, \bigcirc_\chi \mathbf{c}\}]\bot$
4	$\mathbf{b} \gtrdot \mathbf{c} \gtrdot \#$	$\{\mathbf{b}, \bigcirc^{\mathrm{s}}_\chi \mathbf{c}\}$	$[\mathbf{a}, \{\mathbf{a}, \bigcirc_\chi \mathbf{c}\}]\bot$
5	$\mathbf{c} \gtrdot \#$	$\{\mathbf{c}\}$	$[\mathbf{b}, \{\mathbf{b}, \bigcirc^{\mathrm{s}}_\chi \mathbf{c}\}][\mathbf{a}, \{\mathbf{a}, \bigcirc_\chi \mathbf{c}\}]\bot$
6	$\mathbf{c} \gtrdot \#$	$\{\mathbf{c}, \bigcirc^{\mathrm{s}}_\chi \mathbf{c}, \bigcirc^{\mathrm{end}}_\chi \mathbf{c}\}$	$[\mathbf{a}, \{\mathbf{a}, \bigcirc_\chi \mathbf{c}\}]\bot$
7	$\#$	$\{\#\}$	$[\mathbf{c}, \{\mathbf{a}, \bigcirc_\chi \mathbf{c}\}]\bot$
8	$\#$	$\{\#\}$	$\bot$

Figure 4: OPM (left) and a computation (right) on **abbc** of the automaton for formula $\bigcirc_\chi \mathbf{c}$.

$\Phi, \Theta, \Psi \in \mathrm{Atoms}(\varphi)$, $(\Phi, \Theta, \Psi) \in \delta_{pop}$ only if Ψ is the smallest set such that $\Phi \subseteq \Psi$, and for any $\mathrm{p} \in AP$, $\mathrm{p} \in \Phi$ iff $\mathrm{p} \in \Psi$, and containing additional formulas required by the rules detailed in the next paragraphs.

Additionally, for the linear until operator, let $\Phi \in \mathrm{Atoms}(\varphi)$, with $\psi \mathscr{U} \theta \in \Phi$. Then $\psi, \theta, \bigcirc(\psi \mathscr{U} \theta) \in \mathrm{Cl}(\varphi)$, and either $\theta \in \Phi$, or $\psi \in \theta$ and $\bigcirc(\psi \mathscr{U} \theta) \in \Phi$.

Matching Next ($\bigcirc_\chi$) Operator. The $\bigcirc_\chi$ and $\ominus_\chi$ operators are defined w.r.t. maximal chains: the main difficulty posed by their model-checking is building automata able to discern them from inner chains, when multiple chains start in the same position. The automaton for model-checking $\bigcirc_\chi$ relies on the stack in order to keep track of its requirements: if a formula $\bigcirc_\chi \psi$ holds in a state where a chain begins, the automaton stores the auxiliary symbol $\bigcirc^{\mathrm{s}}_\chi \psi$ onto the stack, and pops it when the chain ends, forcing ψ to hold in the appropriate state. More formally, if $\bigcirc_\chi \psi \in \mathrm{Cl}(\varphi)$, then we add $\psi, \bigcirc^{\mathrm{s}}_\chi \psi, \bigcirc^{\mathrm{end}}_\chi \psi \in \mathrm{Cl}(\varphi)$. Also, we enrich the previous constraints on δ as follows: for any $\Phi, \Theta, \Psi \in \mathrm{Atoms}(\varphi)$ and $a \in \mathscr{P}(AP)$, (1) $(\Phi, a, \Theta) \in \delta_{\mathrm{push}} \cap \delta_{\mathrm{shift}}$ iff when $\bigcirc_\chi \psi \in \Phi$, we have $\bigcirc^{\mathrm{s}}_\chi \psi \in \Theta$ and $\bigcirc^{\mathrm{end}}_\chi \psi \notin \Theta$; (2) $(\Phi, a, \Theta) \in \delta_{\mathrm{shift}}$ iff whenever $\bigcirc^{\mathrm{s}}_\chi \psi \in \Phi$, we also have $\psi \in \Phi$, $\bigcirc^{\mathrm{end}}_\chi \psi \in \Phi$, and $\bigcirc^{\mathrm{end}}_\chi \psi \notin \Theta$; $(\Phi, \Theta, \Psi) \in \delta_{pop}$ iff (3) when $\bigcirc^{\mathrm{s}}_\chi \psi \in \Theta$, we have $\bigcirc^{\mathrm{s}}_\chi \psi \in \Psi$, and (4) when $\bigcirc^{\mathrm{s}}_\chi \psi \in \Phi$, then also $\psi \in \Phi$, $\bigcirc^{\mathrm{end}}_\chi \psi \in \Phi$, and $\bigcirc^{\mathrm{end}}_\chi \psi \notin \Psi$. Finally, for any formula $\bigcirc_\chi \psi \in \mathrm{Cl}(\varphi)$, we must exclude from F all atoms containing $\bigcirc^{\mathrm{s}}_\chi \psi$ or $\bigcirc_\chi \psi$.

To explain how the rules above work, we refer to the example computation of the automaton for formula $\bigcirc_\chi \mathbf{c}$ of Fig. 4. $\bigcirc_\chi \mathbf{c}$ holds in the initial state: since the symbol # yields precedence to all other symbols, the first one (namely $a = \{\mathbf{a}\}$) is consumed by a push transition. Due to constraint (1), the auxiliary operator $\bigcirc^{\mathrm{s}}_\chi \psi$ is forced into the next state. If a was not the first symbol, and it had been read by a shift, the same constraint would hold. Since a chain starts in a, the next symbol is also read by a push transition, which stores the previous state, containing $\bigcirc^{\mathrm{s}}_\chi \mathbf{c}$, on the stack (step 2-3 of Fig. 4). Then, the chain ends with the second $\{\mathbf{b}\}$ symbol, and the state mentioned above is popped. The operand of $\bigcirc_\chi$ must hold at the end of the outermost chain, and this is an inner one. However, the automaton still cannot discern between the two cases, and must wait for the next transition to occur, so rule (3) just forces $\bigcirc^{\mathrm{s}}_\chi \mathbf{c}$ into the next state (step 3-4). If the next transition is a push, it means a yields precedence to the symbol read by it, and this is not the outermost chain: the current state, containing $\bigcirc^{\mathrm{s}}_\chi \mathbf{c}$, is again pushed on stack (step 4-5). The process goes on until the pop transition at the end of a chain starting in a is followed by another pop or a shift: this either means a takes precedence from the current symbol, and its stack symbol must be popped, or they are equal in precedence. In both cases, we reached the end of the outermost chain, and ψ must hold in the next state. First, in step 5-6 rule (3) puts $\bigcirc^{\mathrm{s}}_\chi \mathbf{c}$ in the current state: for rule (2) to be satisfied by the shift transition of step 6-7, also $\mathbf{c}$ and $\bigcirc^{\mathrm{end}}_\chi \mathbf{c}$ must hold. If in step 6-7 a pop transition occurred instead of a shift, the same constraints would have been enforced by rule (4). The symbol $\bigcirc^{\mathrm{end}}_\chi$ is another auxiliary operator that marks the satisfaction of the original $\bigcirc_\chi$ formula, and its purpose is to make rule (1) block computations in which a chain does not start in a, but ψ holds

in the next symbol. The $\ominus_\chi$ operator can be treated in an analogous way.

OP Summary Until (Since). For any $O \subseteq \{\lessdot, \doteq, \gtrdot\}$, if $\psi \mathscr{U}^O \theta \in \mathrm{Cl}(\varphi)$ then $\psi, \theta, \bigcirc_\chi(\psi \mathscr{U}^O \theta)$, $\bigcirc(\psi \mathscr{U}^O \theta) \in \mathrm{Cl}(\varphi)$. Also, for any $\Phi \in \mathrm{Atoms}(\varphi)$, $\psi \mathscr{U}^O \theta \in \Phi$ iff either: (1) $\theta \in \Phi$; or (2) $\psi \in \Phi$ and $\bigcirc_\chi(\psi \mathscr{U}^O \theta) \in \Phi$; or (3) $\psi \in \Phi$ and $\bigcirc(\psi \mathscr{U}^O \theta) \in \Phi$. If (1) holds, then $\psi \mathscr{U}^O \theta$ is trivially true; if (2) holds, the path skips the body of a chain starting in the current position, where ψ holds; if (3) holds, then ψ is true in the current position and the path continues in the next one. In the latter case, we must make sure the path is followed only if the current position and the next one are in one of the precedence relations in O. This can be achieved by adding the following constraints: if only (3) holds in a state Φ and $\psi \mathscr{U}^O \theta \in \Phi$, then for any $a, b \in \mathscr{P}(AP)$ and $\Theta \in \mathrm{Atoms}(\varphi)$, with $b = \Theta \cap AP$, we have $(\Phi, a, \Theta) \in \delta_{push} \cup \delta_{shift}$ only if $a \odot b$ and $\odot \in O$. We need not impose constraints on pop transitions because they do not consume input symbols, and they preserve the same subset of AP contained in the starting state. Furthermore, for any $\Phi \in \mathrm{Atoms}(\varphi)$, if $\Phi \in F$ then $\psi \mathscr{U}^O \theta \notin \Phi$, unless $\theta \in \Phi$.

The construction for the since counterpart is analogous: $\bigcirc_\chi$ and $\bigcirc$ are substituted with $\ominus_\chi$ and $\ominus$.

Yield-Precedence Hierarchical Until. This construction exploits the automaton's stack to keep track of the requirements of the $\mathscr{U}^\uparrow$ operator in a way similar to $\bigcirc_\chi$. If $\psi \mathscr{U}^\uparrow \theta \in \mathrm{Cl}(\varphi)$ we introduce the auxiliary operators $\mathscr{U}_{\mathrm{s}}^\uparrow$ and $\mathscr{U}_{\mathrm{end}}^\uparrow$. We add $\psi \mathscr{U}_{\mathrm{s}}^\uparrow \theta, \bigcirc(\psi \mathscr{U}_{\mathrm{s}}^\uparrow \theta), \psi \mathscr{U}_{\mathrm{end}}^\uparrow \theta \in \mathrm{Cl}(\varphi)$, and, for any $\Phi \in \mathrm{Atoms}(\varphi)$ such that $\psi \mathscr{U}^\uparrow \theta \in \Phi$, also $\bigcirc(\psi \mathscr{U}_{\mathrm{s}}^\uparrow \theta) \in \Phi$. Given any $\Phi, \Theta, \Psi \in \mathrm{Atoms}(\varphi)$, we have $(\Phi, \Theta, \Psi) \in \delta_{pop}$ (1) iff $\psi \mathscr{U}_{\mathrm{s}}^\uparrow \theta \notin \Phi$ and $\psi \mathscr{U}_{\mathrm{end}}^\uparrow \theta \notin \Phi$, and if $\psi \mathscr{U}_{\mathrm{s}}^\uparrow \theta \in \Theta$, then either (2) $\theta \in \Psi$ and $\psi \mathscr{U}_{\mathrm{end}}^\uparrow \theta \in \Psi$, or (3) $\psi \in \Psi$ and $\psi \mathscr{U}_{\mathrm{s}}^\uparrow \theta \in \Psi$. Moreover, for any $\Phi, \Theta \in \mathrm{Atoms}(\varphi)$ and $a \in \mathscr{P}(AP)$, (4) if $(\Phi, a, \Theta) \in \delta_{shift}$ then $\psi \mathscr{U}_{\mathrm{s}}^\uparrow \theta, \psi \mathscr{U}_{\mathrm{end}}^\uparrow \theta \notin \Phi$; and (5) if $(\Phi, a, \Theta) \in \delta_{push}$ then $\psi \mathscr{U}_{\mathrm{end}}^\uparrow \theta \notin \Theta$. Moreover, states containing $\psi \mathscr{U}_{\mathrm{s}}^\uparrow \theta$ or $\psi \mathscr{U}_{\mathrm{end}}^\uparrow \theta$ are excluded from the final set F.

Similarly to what happens for $\bigcirc_\chi$, the auxiliary operator $\psi \mathscr{U}_{\mathrm{s}}^\uparrow \theta$ is put into the state after the one in which $\psi \mathscr{U}^\uparrow \theta$ holds, this time by a $\bigcirc$ operator. Then, the state containing the auxiliary symbol is pushed on stack. Each time it is popped, we check if either θ holds in the next state (rule (2)), and $\psi \mathscr{U}_{\mathrm{end}}^\uparrow \theta$ is put into the next state to mark the end of the hierarchical path, or if ψ holds and the path continues, so $\psi \mathscr{U}_{\mathrm{s}}^\uparrow \theta$ must continue being propagated. Let j be the ending position of the maximal chain starting where $\psi \mathscr{U}^\uparrow \theta$ is supposed to hold. The purpose of rules (1) and (4) is to stop the computation if the hierarchical path does not contain a position before j in which θ holds (recall that j is not part of the hierarchical path, only non-maximal chain ends are).

Other Hierarchical Operators. The automata recognizing other hierarchical operators can be built similarly. In particular, the automaton for $\mathscr{S}^\downarrow$ is similar to the one for $\mathscr{U}^\uparrow$, while those for $\mathscr{U}^\downarrow$ and $\mathscr{S}^\uparrow$ formulas are closer to the one recognizing $\ominus_\chi$.

Complexity. The size of set $\mathrm{Cl}(\varphi)$ is linear w.r.t the length of formula φ, and set $\mathrm{Atoms}(\varphi)$, which contains the states of the automaton, is a subset of $\mathscr{P}(\mathrm{Cl}(\varphi))$, being of size exponential in $\mathrm{Cl}(\varphi)$. Therefore, we can state that for any OPTL formula φ it is possible to build an OPA of size $2^{O(|\varphi|)}$ that accepts models satisfying φ. Note that, despite translating hierarchical operators into summary operators yields an exponential blow-up of formula length (see Section 3.2), they do not lead to an increase in the complexity of model checking. This is in contrast with the behavior of the within operator in NWTL, which exponentially increases model checking complexity.

5.1 Hints for Model Checking Infinite Words

Model checking for an OPTL formula φ on infinite words can be performed by building a generalized OP Büchi Automaton (ωOPBA, [17]) $\mathscr{A}_\varphi^\omega = \langle \mathscr{P}(AP), M_{\mathscr{P}(AP)}, \mathrm{Atoms}(\varphi), I, \mathbf{F}, \delta \rangle$, which differs from the finite-word counterpart only for the acceptance condition. $\mathbf{F}$ is the set of sets of Büchi final states, and an

ω-word is accepted iff at least one state from each one of the sets contained in $\mathbf{F}$ is visited infinitely often during the computation. The main difficulty in adapting the finite-word construction to the infinite case is the acceptance condition. In finite words, the stack is empty at the end of every accepting computation, which implies all temporal constraints tracked by stack symbols must have been satisfied. In ωOPBAs, the stack may never be empty, and symbols containing auxiliary operators may remain buried forever, never enforcing the satisfaction of the formulas they refer to. This problem can be solved by defining additional auxiliary operators for each temporal formula that requires stack support, such as $\bigcirc_\chi$, $\ominus_\chi$ and hierarchical operators. Suppose we want to model check $\bigcirc_\chi \psi$. One of these new symbols, $\bigcirc^{\mathrm{p}}_\chi \psi$, must be inserted in the current state whenever $\bigcirc^{\mathrm{s}}_\chi \psi$ is pushed on stack, and kept in the automaton's state until $\bigcirc^{\mathrm{s}}_\chi \psi$ is popped, and its temporal requirement satisfied. When this happens another symbol, say $\bigcirc^{\omega\mathrm{end}}_\chi \psi$, must be placed into the current state to mark satisfaction, and removed afterwards. Then, it is possible to define an acceptance set $F \in \mathbf{F}$ for each use of this temporal operator. F only contains atoms that either contain $\bigcirc^{\omega\mathrm{end}}_\chi \psi$ (so the formula can be satisfied infinitely often), or atoms not containing $\bigcirc^{\mathrm{p}}_\chi \psi$ (so words where $\bigcirc_\chi \psi$ needs not be satisfied infinitely often are accepted). When adding constraints on the transition relation to realize this behavior, particular care must be taken in order to prevent $\bigcirc^{\omega\mathrm{end}}_\chi \psi$ form remaining when it should not, and not to remove $\bigcirc^{\mathrm{p}}_\chi \psi$ too early when multiple instances of the same subformula are pending, and only one of them is satisfied.

6 Conclusions

We presented a new temporal logic based on the OPL family. We proved that it is more expressive than NWTL, which is based on the less powerful class VPL. OPTL can express more properties than other similar temporal logics, with the possibility of model checking with an automaton of size not greater than competing formalisms. A further natural step in the theoretical characterization of OPTL is a more complete exploration of its expressiveness, in particular concerning First-Order completeness, as it has been done for various temporal logics, including NWTL [1].

On the side of practical application we are planning the development and implementation of suitable model checking algorithms exploiting the theoretical procedures presented in this paper. We also believe that "core temporal logic languages", such as LTL, CTL, CTL*, and, even more, those derived therefrom to attack subclasses of CFLs, including NWTL and OPTL, lack user-friendliness and require excessive mathematical skill to translate an informal requirement into a well-defined formula; thus, we envisage a higher level interface for OPTL more palatable for users familiar with semi-formal notations like UML.

References

[1] R. Alur, M. Arenas, P. Barceló, K. Etessami, N. Immerman & L. Libkin (2008): *First-Order and Temporal Logics for Nested Words*. Logical Methods in Computer Science 4(4), doi:10.2168/LMCS-4(4:11)2008.

[2] R. Alur, S. Chaudhuri & P. Madhusudan (2011): *Software model checking using languages of nested trees*. ACM Trans. Program. Lang. Syst. 33(5), pp. 15:1–15:45, doi:10.1145/2039346.2039347.

[3] R. Alur & D. L. Dill (1994): *A Theory of Timed Automata*. Theor. Comput. Sci. 126(2), pp. 183–235, doi:10.1016/0304-3975(94)90010-8.

[4] R. Alur & P. Madhusudan (2009): *Adding nesting structure to words*. JACM 56(3), doi:10.1145/1516512.1516518.

[5] A. Barenghi, S. Crespi Reghizzi, D. Mandrioli, F. Panella & M. Pradella (2015): *Parallel parsing made practical*. Sci. Comput. Program. 112, pp. 195–226, doi:10.1016/j.scico.2015.09.002.

[6] A. Bouajjani, J. Esparza & O. Maler (1997): *Reachability analysis of pushdown automata: Application to model-checking*. In: *CONCUR '97: Concurrency Theory*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 135–150, doi:10.1007/3-540-63141-0_10.

[7] B. von Braunmühl & R. Verbeek (1983): *Input-driven languages are recognized in log n space*. In: *Proc. of the Symp. on Fundamentals of Computation Theory, LNCS 158*, Springer, pp. 40–51, doi:10.1007/3-540-12689-9_92.

[8] J. R. Büchi (1960): *Weak Second-Order Arithmetic and Finite Automata*. *Mathematical Logic Quarterly* 6(1-6), pp. 66–92, doi:10.1002/malq.19600060105.

[9] O. Burkart & B. Steffen (1992): *Model checking for context-free processes*. In: *CONCUR '92, LNCS* 630, Springer Berlin Heidelberg, pp. 123–137, doi:10.1007/BFb0084787.

[10] S. Crespi Reghizzi & D. Mandrioli (2012): *Operator Precedence and the Visibly Pushdown Property*. *JCSS* 78(6), pp. 1837–1867, doi:10.1016/j.jcss.2011.12.006.

[11] S. Crespi Reghizzi, D. Mandrioli & D. F. Martin (1978): *Algebraic Properties of Operator Precedence Languages*. *Information and Control* 37(2), pp. 115–133, doi:10.1016/S0019-9958(78)90474-6.

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

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

[14] M. Frick & M. Grohe (2004): *The complexity of first-order and monadic second-order logic revisited*. *Ann. Pure Appl. Logic* 130(1-3), pp. 3–31, doi:10.1016/j.apal.2004.01.007.

[15] D. Grune & C. J. Jacobs (2008): *Parsing techniques: a practical guide*. Springer, New York, doi:10.1007/978-0-387-68954-8.

[16] C. Lautemann, T. Schwentick & D. Thérien (1994): *Logics For Context-Free Languages*. In: *Computer Science Logic, 8th International Workshop, CSL '94*, pp. 205–216, doi:10.1007/BFb0022257.

[17] V. Lonati, D. Mandrioli, F. Panella & M. Pradella (2015): *Operator Precedence Languages: Their Automata-Theoretic and Logic Characterization*. *SIAM J. Comput.* 44(4), pp. 1026–1088, doi:10.1137/140978818.

[18] D. Mandrioli & M. Pradella (2018): *Generalizing input-driven languages: Theoretical and practical benefits*. *Computer Science Review* 27, pp. 61–87, doi:10.1016/j.cosrev.2017.12.001.

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

[20] R. McNaughton & S. Papert (1971): *Counter-free Automata*. MIT Press, Cambridge, USA.

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

[22] I. Walukiewicz (2001): *Pushdown Processes: Games and Model-Checking*. *Information and Computation* 164(2), pp. 234–263, doi:10.1006/inco.2000.2894.

[23] P. Wolper, M. Y. Vardi & A. P. Sistla (1983): *Reasoning about infinite computation paths*. In: *24th Annual Symposium on Foundations of Computer Science SFCS*, pp. 185–194, doi:10.1109/SFCS.1983.51.