Theoretical Computer Science 848 (2020) 47–81

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Operator precedence temporal logic and model checking ☆,☆☆

Michele Chiari [a,*], Dino Mandrioli [a], Matteo Pradella [a,b]

[a] DEIB, Politecnico di Milano – P.zza L. Da Vinci, 32, 20133, Milano, Italy
[b] IEIIT, Consiglio Nazionale delle Ricerche, Italy

ARTICLE INFO

Article history:
Received 8 April 2019
Received in revised form 11 August 2020
Accepted 31 August 2020
Available online 3 September 2020
Communicated by P. Aziz Abdulla

Keywords:
Operator precedence languages
Visibly pushdown languages
Input driven languages
ω-Languages
Temporal logic
Model checking

ABSTRACT

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. Our results cover both finite and infinite string languages.

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* (MSO) logic [2] and the definition of a number of subfamilies that are all equivalent between each other and are characterized in terms of its first-order fragment (FO) [3].

☆ Work partially supported by project AUTOVAM, funded by Fondazione Cariplo and Regione Lombardia, grant n. 2017-1213.
☆☆ A preliminary version of this work was presented at the conference GandALF 2018 [1].
* Corresponding author.
E-mail addresses: michele.chiari@polimi.it (M. Chiari), dino.mandrioli@polimi.it (D. Mandrioli), matteo.pradella@polimi.it (M. Pradella).

https://doi.org/10.1016/j.tcs.2020.08.034
0304-3975/

Such foundational results, however, remained of essentially theoretic interest because the formal correctness problem, though decidable, remains of intractable complexity for MSO and FO logics (see e.g., [4]). The state of the art, however, had a dramatic breakthrough with the advent of *temporal logic and model checking* [5]: 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 [6], 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 markup languages such as XML and HTML. 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 [7], which correspond to regular tree-languages [8]; among various extensions thereof special attention have received *input-driven* (ID) [9], alias *visibly push-down* languages (VPLs) [10]. 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 [10]; early attempts have also been done to support their model checking by exploiting suitable extensions of temporal logics –whether linear time [11,12] or branching time [13].

It is also worth mentioning some earlier results on model checking state machines whose nondeterministic computations have a typical tree-shaped structure by means of a variant of alternation-free modal μ-calculus (a branching time logic) [14], whose relationship with pushdown games is discussed in [15]. Moreover, the problem of model checking regular properties on pushdown systems has been the object of a thorough exploration. Reachability analysis of pushdown systems with alternating automata has been studied in [16]. Techniques for model checking against pushdown systems, w.r.t. both linear and branching time, have been given in [17–20]. In these works, specifications are given by means of temporal logics such as LTL, CTL and variations thereof, limiting the scope to regular properties only. An early attempt at embracing a wider class of properties was made in [21], where specifications were given in a decidable restriction of a Presburger logic system. Later, [22] studied model checking of context-free specifications, expressed with tree automata, against regular system models.

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 [23]; subsequently, we showed that they enjoy many of the algebraic properties of structured CFLs [24]; 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 [26] 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. Furthermore, unlike regular languages and VPLs, OPLs are not necessarily real-time.

We have built an MSO characterization of OPLs [27] by extending in a non-trivial way the key relation between "matching string positions" introduced in [28] and exploited in [10] 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 [29].

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. The extensive literature on VPLs shows that there is, in fact, the need for such results. In particular, VPLs have been introduced with the purpose of model checking context-free properties of procedural programs [11]. However, as it is shown also in [30], they are not expressive enough to model programs with exceptions, a construct that is ubiquitous in modern programming languages, and similar advanced control mechanisms, such as continuations, generators, coroutines, and so on. As we shall see in the next sections, OPLs are well-suited to model them, thus closing an important gap in the current state of the art. Many other systems that cannot be modeled by VPLs can be modeled by OPLs, enabling their model checking, e.g. interrupt-driven software, version-control systems, and database logs with rollbacks [27]. Moreover, OPLs have been shown to be expressive enough to model some popular data-representation languages, such as XML and JSON [25]. Thus, the development of temporal logics based on them enables reasoning on data stored in such languages, taking into account their hierarchical structure, by means of query evaluation. This problem has already been studied for XML in the VPL context with nested words [31], but OPLs extend such results by enabling modeling of XML documents with unmatched tags, and HTML.

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 [12], but requires many more technicalities due to the lack of the "matching relation" [28] typical of parenthesis languages. We formally define OPTL syntax and semantics for finite words and provide examples of its usage and its generality. In

[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 [25], but this issue is not the object of the present paper.

	call	ret	han	thr
call	$\lessdot$	$\doteq$	$\lessdot$	$\gtrdot$
ret	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
han	$\lessdot$	$\gtrdot$	$\lessdot$	$\lessdot$
thr	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$

Fig. 1. OPM $M_{\mathbf{call}}$.

Section 4 we show that OPTL defines a larger language family than [12]'s NWTL: every NWTL formula 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. Then, in Section 6 we extend OPTL to ω-words, by highlighting the differences with the finite case. Afterwards, in Section 6 we show how to extend to the infinite case the model checking procedure we devised in Section 5. Section 7 concludes and envisages several further research steps.

2. Operator precedence languages and automata

Operator Precedence languages (OPL) are normally defined through their generating grammars [23]; in this paper, however, we characterize them through their accepting automata [27], which are the natural way to state equivalence properties with logic characterizations. We assume some familiarity with classical language theory concepts such as context-free grammar, parsing, shift-reduce algorithms, syntax trees [32].

Let Σ be an alphabet. The empty string is written 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 [32]. The context-free structure of words in OPLs is given by precedence relations between pairs of input symbols, which are defined in a operator precedence matrix.

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 $M_{a,b} = \odot$, where $\odot \in \{\lessdot, \doteq, \gtrdot\}$, we write $a \odot b$. For $u, v \in \Sigma^+$ we write $u \odot v$ if $u = xa$ and $v = by$ with $a \odot b$.

An OPM completely determines the context-free structure of words on the same alphabet through the concept of *chain*, which is essential to the logical characterizations of OPLs.

Definition 2.2. A *simple chain* is a string $c_0c_1c_2 \dots c_\ell c_{\ell+1}$, written as

$$^{c_0}[c_1c_2 \dots c_\ell]^{c_{\ell+1}}$$

such that: $c_0, c_{\ell+1} \in \Sigma \cup \{\#\}$, $c_i \in \Sigma$ for every $i = 1, 2, \dots \ell$ $(\ell \geq 1)$, 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 $c_0s_0c_1s_1c_2 \dots c_\ell s_\ell c_{\ell+1}$, where $^{c_0}[c_1c_2 \dots c_\ell]^{c_{\ell+1}}$ 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 $i = 0, 1, \dots, \ell$ $(\ell \geq 1)$. Such a composed chain will be written as $^{c_0}[s_0c_1s_1c_2 \dots c_\ell s_\ell]^{c_{\ell+1}}$. Each composed chain has an underlying simple chain.

The first symbol of a chain is called its *left context*, and the last one its *right context*. All characters in between are called its *body*.

Definition 2.3. 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.

An OPM uniquely determines the way a compatible word is structured into chains. A complete OPM on Σ determines a unique way to structure any word in Σ^* into chains since all words are compatible with it.

Example. OPM $M_{\mathbf{call}}$ is shown in Fig. 1. It is conceived to model the stack trace of a computer program, containing function calls, denoted by the label **call**, and returns (**ret**). Functions may raise exceptions with a **thr** statement, and they may also catch exceptions by installing handlers (**han**). A single handler may catch any number of exceptions.

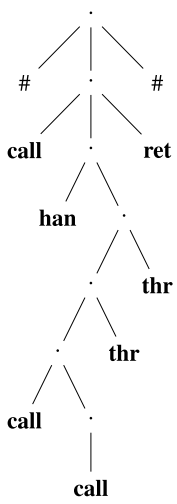


Fig. 2. AST of w_{ex} w.r.t. OPM $M_{\mathbf{call}}$. Dots are non-terminal symbols.

In order to see how the syntactic structure of a sentence is determined by the OPM and is formalized by the concept of chain, let us apply the traditional operator precedence parsing algorithm to the sample word $w_{ex} =$ **call han call call thr thr ret** (for a more complete treatment, cf. [29,32]). First, write all precedence relations between consecutive characters, according to OPM $M_{\mathbf{call}}$. Then, recognize all innermost patterns of the form $a \lessdot c \doteq \cdots \doteq c \gtrdot b$ as simple chains, and remove their bodies. In the parsing setting, a chain body is composed by the terminal symbols of the right hand side of a grammar rule, so removing it can be seen as a reduction. Then, write the precedence relations between the left and right contexts of the removed body, a and b, and iterate this process until only # characters remain. This procedure is applied to w_{ex} as follows:

0 | # ⋖ **call** ⋖ **han** ⋖ **call** ⋖ **<u>call</u>** ⋗ **thr** ⋗ **thr** ⋗ **ret** ⋗ #
1 | # ⋖ **call** ⋖ **han** ⋖ **<u>call</u>** ⋗ **thr** ⋗ **thr** ⋗ **ret** ⋗ #
2 | # ⋖ **call** ⋖ **han** ⋖ **<u>thr</u>** ⋗ **thr** ⋗ **ret** ⋗ #
3 | # ⋖ **call** ⋖ **han** ⋖ **<u>thr</u>** ⋗ **ret** ⋗ #
4 | # ⋖ **call** ⋖ **<u>han</u>** ⋗ **ret** ⋗ #
5 | # ⋖ **<u>call</u>** ≐ **<u>ret</u>** ⋗ #
6 | # ≐ #

The chain body removed in each step is underlined. In step 0, $^{\mathbf{call}}[\underline{\mathbf{call}}]^{\mathbf{thr}}$ is a simple chain, so its body **<u>call</u>** is removed. Then, in step 1 we recognize the simple chain $^{\mathbf{han}}[\underline{\mathbf{call}}]^{\mathbf{thr}}$, which means $^{\mathbf{han}}[\mathbf{call}[\mathbf{call}]]^{\mathbf{thr}}$, where [**call**] is the chain body removed in step 0, is a composed chain. This way, we recognize, e.g., $^{\mathbf{call}}[\mathbf{call}]^{\mathbf{thr}}$, $^{\mathbf{han}}[\mathbf{thr}]^{\mathbf{ret}}$ as simple chains, and $^{\mathbf{call}}[\mathbf{call}[\mathbf{call}]]^{\mathbf{thr}}$ and $^{\mathbf{han}}[[[\mathbf{call}[\mathbf{call}]]\mathbf{thr}]\mathbf{thr}]^{\mathbf{ret}}$ as composed chains (with inner chain bodies enclosed in brackets). Overall, w_{ex} is structured by the following composed chain:

#[**call**[**han**[[[**call**[**call**]]**thr**]**thr**]]**ret**]#

which is an isomorphic representation of w_{ex}'s abstract syntax tree (AST), where nonterminal symbols are omitted, as depicted in Fig. 2. The distinguishing feature of OPLs, in fact, is that during their parsing nonterminals are "transparent". This justifies labeling them as *input-driven but not visible* push-down languages ([29]).

Notice how each chain body corresponds to the expansion of one non-terminal (represented by a dot in the figure). The left (right) context of a chain is the first terminal to the left (right) of such non-terminal. Two terminals that are consecutive –or separated by a non-terminal– are in the ⋖ relation iff in the AST the second one is on a lower level than the first one, in the ⋗ relation iff the first one is on a lower level, and in the ≐ relation iff they are on the same level.

According to OPM $M_{\mathbf{call}}$, each **call** position, together with the subcalls issued by the same procedure, is enclosed in a chain body between the **call** or the **han** after which they are initiated, and the statement terminating their procedures. This can both be a **ret**, or a **thr** raising an uncaught exception. Each **ret** terminates exactly one call, generating a model similar to nested words [10]. **thr** statements may, instead, terminate multiple procedures. This can be expressed in our model, because a position may be the context of multiple chains. In w_{ex}, the first **thr** terminates multiple **call**s. So, it is the right context of the chains whose bodies contain such calls. Note that this many-to-one relation cannot be modeled with nested words, because their matching relation is strictly one-to-one, with the only exception of unmatched **ret** at the beginning and **call** at the end of the sentence. The greater expressive power of OPL w.r.t. VPL, the language class upon which nested words are based, is further detailed in [29].

OPMs, therefore, provide a unique, implicit structure to a universe of strings of Σ^* – which is exactly Σ^* if the OPM is complete. We now present the class of automata that defines OPLs for finite strings, and later extend it to ω-words. These automata are driven by the precedence relations defined in the OPM, which determine the automaton's moves according to the chain structure. They allow us to define languages that are subsets of the set of words compatible with a given OPM.

Definition 2.4. 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 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:

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

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

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 use the notation $q_0 \xrightarrow{a} q_1$ for $(q_0, a, q_1) \in \delta_{\text{push}}$, $q_0 \overset{a}{\dashrightarrow} q_1$ for $(q_0, a, q_1) \in \delta_{\text{shift}}$, $q_0 \overset{q_2}{\Longrightarrow} q_1$ 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 $(q, p, r) \in \delta_{\text{pop}}$.

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 $L(\mathcal{A}) = \{x \in \Sigma^* \mid \langle x\#,\ q_I,\ \bot \rangle \vdash^* \langle \#,\ q_F,\ \bot \rangle, q_I \in I, q_F \in F\}$.

The portion of an OPA run driven by a chain in the input string is called a *support*.

Definition 2.5. Let $\mathcal{A}$ be an OPA. We call a *support* for the simple chain ${}^{c_0}[c_1 c_2 \ldots c_\ell]^{c_{\ell+1}}$ any path in $\mathcal{A}$ of the form $q_0 \xrightarrow{c_1} q_1 \dashrightarrow \ldots \dashrightarrow q_{\ell-1} \overset{c_\ell}{\dashrightarrow} q_\ell \overset{q_0}{\Longrightarrow} q_{\ell+1}$, where the solid arrow corresponds to a push move, whereas the dashed 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* ${}^{c_0}[s_0 c_1 s_1 c_2 \ldots c_\ell s_\ell]^{c_{\ell+1}}$ any path in $\mathcal{A}$ of the form

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

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 ${}^{c_i}[s_i]^{c_{i+1}}$, i.e., it can be decomposed as $q_i \overset{s_i}{\rightsquigarrow} q''_i \overset{q_i}{\Longrightarrow} q'_i$.
- if $s_i = \varepsilon$, then $q'_i = q_i$.

Notice that the label of the last pop is exactly q'_0.

The chains, and thus the OPM, 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

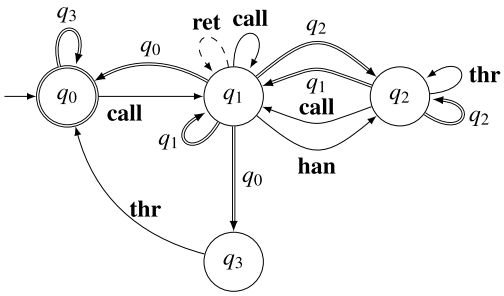


Fig. 3. OPA recognizing the language of program execution traces. Push and shift transitions are represented with, resp., solid and dashed edges, labeled with the terminal symbol they read; pop transitions with double edges, labeled with the state in the stack symbol they pop.

support like (1) with $s = s_0 c_1 \dots c_\ell s_\ell$ and $q_{\ell+1} = q_k$. The above computation corresponds to the parsing by the automaton of the string $s_0 c_1 \dots c_\ell s_\ell$ 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*.

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

For a max-automaton $\mathcal{A}(\Sigma, M)$ each chain has a support; since there is a chain $^{\#}[s]^{\#}$ for any string s compatible with M, a string is accepted by $\mathcal{A}(\Sigma, M)$ 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.

Example (cont.). The max-automaton for $M_{\mathbf{call}}$ accepts all strings on its alphabet, as it gives every chain a support. It is guided by precedence relations, which determine the kind of moves it performs each time. The first character of the body of a simple chain is always read by a push, starting its support, and a pop is always performed after the last one, ending the support. All other characters (in this case only **ret**) are read by a shift, which updates the stack symbol pushed with the first one. Thus, the sequence of moves of an OPA on a given word is determined by the OPM only, and OPA may only differ in word acceptance by enabling or forbidding certain transitions, depending on the current state and, only for pop moves, on the topmost stack symbol.

The max-automaton gives to string $w_{wr} =$ **ret call han** the structure #[[**ret**]**call**[**han**]]#. The initial **ret** can appear without a previous **call** because $\# \lessdot a$, for any $a \in \Sigma$, and the unmatched **call** can too, symmetrically. This is similar to unmatched calls and returns in VPLs.

In Fig. 3, instead, we show an OPA on $M_{\mathbf{call}}$ that only accepts program execution traces that are well-formed according to the following rules: all **call**s are matched either singularly by a **ret** or multiply by a **thr**; a **thr** pops all pending **call**s until a handler is found, if any; otherwise the stack is completely emptied; a single handler may catch several **thr**s but is uninstalled by a return that matches the **call** that installed it. Neither unmatched **ret**s nor **thr**s outside of a **call** are accepted.

To guarantee the above rules the OPA of Fig. 3 behaves as follows: in state q_0, no function is currently active, so only **call**s can be read. In state q_1, at least one function body is active, so other functions can be called, or a handler can be installed, in which case the OPA goes to state q_2. If a **thr** statement is encountered, all stack symbols pushed with the previous calls are popped, until the one pushed from state q_1 leads to q_2 –a handler was installed– or the one pushed from state q_0 leads to q_3 –no handler is on the stack–, from which **thr** is read with a push and immediately popped without changing the state.

Fig. 4 shows a run of the OPA on w_{ex}. The reader can easily verify that the OPA rejects w_{wr}, as well as any word where a **thr** occurs when the stack is empty.

Note that OPAs are not *real-time*, since pop moves are essentially ε-moves, and use the input symbol as a look-ahead, without actually reading it. In our example, before reading the first **thr** statement of w_{ex}, the OPA pops the stack symbols it pushed when reading the **call**s terminated by it. Thus, a single character (**thr**) ends the multiple supports of the corresponding chains, and also starts a new one. Conversely, VPLs are recognized by automata that are strictly real-time, i.e. they consume exactly one input symbol for each transition. This property determines their more limited expressive power.

Closure properties. OPLs enjoy several important closure properties [24,26]. They are closed under union, complement and intersection, so that OPLs sharing the same OPM form a Boolean algebra. This determines the decidability of the inclusion problem for OPLs, while that of the emptiness problem holds for the whole class of context-free languages. They are also closed w.r.t. concatenation, Kleene star, and reversal. Deterministic OPAs are expressively equivalent to the nondeterministic ones, in contrast to what happens for generic pushdown automata. OPLs have also been characterized by means of monadic second order logic [27].

input	state	stack	relation	next move
call han call call thr thr ret #	q_0	$\bot$	# $\lessdot$ **call**	push
han call call thr thr ret #	q_1	[**call**, q_0]$\bot$	**call** $\lessdot$ **han**	push
call call thr thr ret #	q_2	[**han**, q_1][**call**, q_0]$\bot$	**han** $\lessdot$ **call**	push
call thr thr ret #	q_1	[**call**, q_2][**han**, q_1][**call**, q_0]$\bot$	**call** $\lessdot$ **call**	push
thr thr ret #	q_1	[**call**, q_1][**call**, q_2][**han**, q_1][**call**, q_0]$\bot$	**call** $\gtrdot$ **thr**	pop
thr thr ret #	q_1	[**call**, q_2][**han**, q_1][**call**, q_0]$\bot$	**call** $\gtrdot$ **thr**	pop
thr thr ret #	q_2	[**han**, q_1][**call**, q_0]$\bot$	**han** $\lessdot$ **thr**	push
thr ret #	q_2	[**thr**, q_2][**han**, q_1][**call**, q_0]$\bot$	**thr** $\gtrdot$ **thr**	pop
thr ret #	q_2	[**han**, q_1][**call**, q_0]$\bot$	**han** $\lessdot$ **thr**	push
ret #	q_2	[**thr**, q_2][**han**, q_1][**call**, q_0]$\bot$	**thr** $\gtrdot$ **ret**	pop
ret #	q_2	[**han**, q_1][**call**, q_0]$\bot$	**han** $\gtrdot$ **ret**	pop
ret #	q_1	[**call**, q_0]$\bot$	**call** $\doteq$ **ret**	shift
#	q_1	[**ret**, q_0]$\bot$	**ret** $\gtrdot$ #	pop
#	q_0	$\bot$	# $\doteq$ #	–

Fig. 4. Run of the OPA of Fig. 3 on w_{ex}.

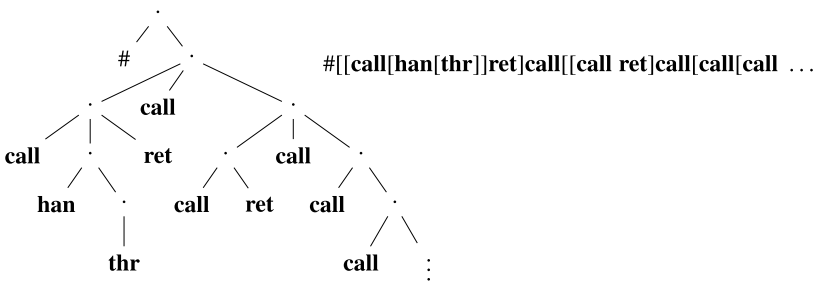


Fig. 5. Example right recursive OP ω-word (right), with its AST (left), on OPM $M_{\textbf{call}}$. The substring [**call** at the end is repeated infinitely many times.

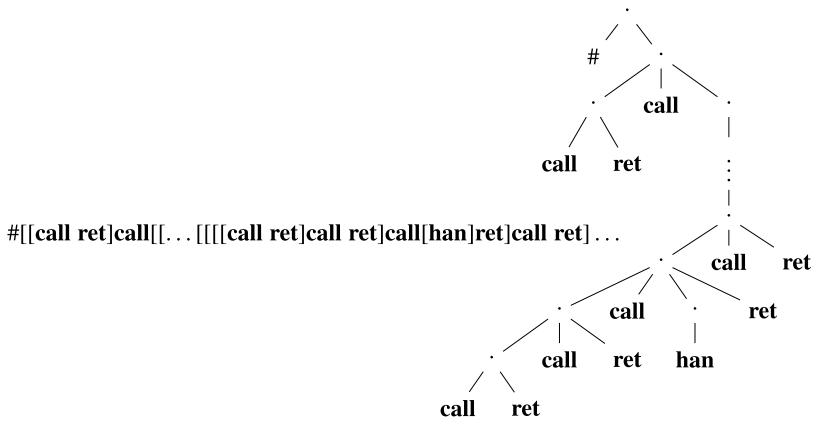


Fig. 6. Example left recursive OP ω-word (left), with its AST (right), on OPM $M_{\textbf{call}}$. The substring **call ret**] at the end is repeated infinitely many times.

2.1. Operator precedence ω-languages

All above definitions are extended to infinite words in the traditional way, but with the following distinctions. 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. Notice that a composed open chain may contain both open and closed subchains, obviously with the constraint that 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.

Example. Figs. 5 and 6 are two examples of OP ω-words. Fig. 5 is right recursive, in the sense that the infinite part of the AST grows toward the right. Thus, each one of the **call**s with no matched **ret** or **thr** is the right context of a different open

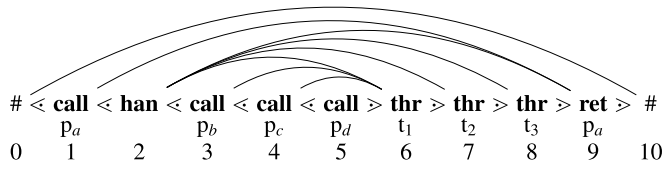


Fig. 7. An example of execution trace (top), according to $M_{\mathbf{call}}$ (Fig. 1). Chains are highlighted by arcs joining their contexts; structural labels are typeset in bold, while other atomic propositions are shown below them. The chain structure is also shown with brackets (bottom). First, procedure p_a is called (position 1), and it installs an exception handler in 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.

chain, and they are all pending positions. Fig. 6 is left recursive, as the tree grows toward the left. The second **call** is the right context of infinitely many chains, and it is the only pending position besides #. Fig. 6 is a bit less intuitive because the tree grows by adding nodes at the root of a subtree, as denoted by the vertical dots.

OPA classes accepting the whole class of ωOPLs can be defined by augmenting Definition 2.4 with Büchi or Muller acceptance conditions [27]. In this paper, we only consider the former. 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 $\mathrm{Inf}(\rho) = \{q \in Q \mid$ there exist infinitely many positions i 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 $L(\mathcal{A}) = \{w \in \Sigma^\omega \mid \mathcal{A}$ 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.

Example (cont.). We invite the reader to simulate (part of) a run ρ of the OPA of Fig. 3 on the words of Figs. 5 and 6 to get familiar with how ωOPBAs work. For both words we have $\mathrm{Inf}(\rho) = \{q_1\}$. With the final set $F = \{q_0\}$, both words are rejected, as the OPA accepts the language of stack traces with all **call**s matched with a **ret** or a **thr**. Changing it to $F = \{q_1, q_2\}$ yields an OPA accepting the language of well-formed stack traces, with unmatched (or pending) **call**s representing procedures that run indefinitely. This kind of behavior is typical of software that runs forever, such as servers, certain embedded systems, and so on.

The most important closure properties of OPLs are preserved by ωOPLs, which still form a Boolean algebra and are closed under concatenation of an OPL with an ωOPL [27]. The equivalence between deterministic and nondeterministic automata is instead lost in the infinite case, which is unsurprising, since it also happens for regular ω-languages and ωVPLs.

To summarize, given an OP alphabet, its OPM M assigns an unambiguous structure to any compatible string in Σ^* (or Σ^ω); 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 (or an ωOPBA) 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.

3. Operator precedence temporal logic

As we remarked in the previous section, languages recognized by different OPAs on a given OP alphabet form a Boolean algebra. These properties allow us to define OPTL (*Operator Precedence Temporal Logic*) as a sound propositional temporal logic, with logical disjunction, conjunction and negation. 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, we formally define its semantics on finite words, and provide more complex, real-world examples in Section 3.3. We extend its definition to infinite words in Section 6.

3.1. Syntax and informal semantics

Let AP be the finite set of atomic propositions: 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 are typeset in bold face. 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, in the word of Fig. 7, position 1 is labeled with the set $\{\mathbf{call}, \mathrm{p}_a\}$, and position 3 with $\{\mathbf{call}, \mathrm{p}_b\}$: they both contain **call**, so they are in the $\lessdot$ relation according to matrix $M_{\mathbf{call}}$ of Fig. 1.

The syntax of OPTL is given by the following grammar, where 'a' denotes any symbol in AP:

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

We informally show the meaning of OPTL operators by referring to the word of Fig. 7, w.r.t. the OPM of Fig. 1. The $\bigcirc$ and $\ominus$ symbols denote the next and back operators from LTL, while the undecorated $\mathcal{U}$ and $\mathcal{S}$ operators are LTL until and since. These operators have the same semantics as in LTL, and thus they may only express regular properties, although they do so on OP words, whose structure is tree-like, and not only regular. However, in order to fully exploit the expressive power of OPLs, operators capable of interacting with the peculiarities of their structure are needed.

The $\bigcirc_\chi$ and $\ominus_\chi$ operators, which we call *matching next* and *matching back*, express properties on string positions in the maximal chain relation (which will be formally defined later on) with the current one. In the figure, the chain relation between two positions is shown by an arc joining them. A chain is maximal if it is the outermost one starting or ending in a position: for example, the chain between positions 2 and 9 is maximal, while the one between 2 and 8 is not, because it is contained in the body of the former. For example, formula $\bigcirc_\chi$ **thr**, when evaluated in positions containing a **call**, is true if the corresponding procedure is terminated by an exception thrown by an inner procedure, such as 3 and 4 of Fig. 7, because position 3 forms a maximal chain with 6, in which **thr** holds, and so on. Formula $\ominus_\chi$ **han**, if evaluated in **thr** positions, is true if the corresponding exception is caught by a **han** statement, such as 6, 7 and 8, because e.g. position 2 forms a chain with 6, and **han** holds in 2.

The $\mathcal{U}^{\boxdot}$ and $\mathcal{S}^{\boxdot}$ operators, called *operator precedence summary until and since*, are inspired to the homonymous $\mathcal{U}^{\sigma}$ and $\mathcal{S}^{\sigma}$ operators from [12], 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. $\mathcal{U}^{\lessdot\doteq}$ or $\mathcal{U}^{\gtrdot\lessdot}$ and so on).

The presence or absence of these relations influences the ability of the *summary* paths associated with these operators to "cross" chains, entering them from the outside or exiting them from the inside; e.g., the $\gtrdot$ relation allows paths starting inside the body of a chain to exit it, and proceed with its right context. Note that the position before the right context of a chain is always in the $\gtrdot$ relation with it. For example, formula (**call** $\vee$ **thr**) $\mathcal{U}^{\gtrdot}$ **ret** is true in position 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: positions 3 and those from 6 to 8 satisfy **call** $\vee$ **thr**, while position 9 satisfies **ret**. Since **call** positions yield precedence to other **call** positions, if this operator is evaluated in such positions, its paths cannot enter the body of an inner procedure call, but they can only jump to the **thr** statement.

The $\lessdot$ relation allows paths to enter call bodies: formula (**han** $\vee$ **call**) $\mathcal{U}^{\lessdot}$ p_d is true in position 2 because of path 2-3-4-5, which enters all chains it encounters. Such a path could, however, skip the maximal chain starting in 2 and continue after 9. This can be prevented by adding $\neg$**ret**, reducing the scope of the formula to a single stack frame: ($\neg$**ret** $\wedge$ (**han** $\vee$ **call**)) $\mathcal{U}^{\lessdot}$ p_d holds in **han** positions that can catch exceptions thrown by a **call** to procedure p_d.

The $\doteq$ relation allows paths to connect consecutive positions that are part of the same right-hand side. This is useful especially in the presence of OPMs with $\doteq$-circularity, such as the one used in Section 4. The precedence relations can be combined together to sum these three different behaviors for the summary until and since operators, posing or lifting restrictions on the way these operators navigate the tree-like structure of OP words. In a sense, $\mathcal{U}^{\lessdot\doteq}$ resembles the summary-down until of [12], while $\mathcal{U}^{\doteq\gtrdot}$ behaves similarly to the summary-up, with the difference that OPTL operators can interact with multiple chains ending in the same position. Similar considerations can be made for the since versions of these operators.

$\mathcal{U}^{\circledcirc}$ and $\mathcal{S}^{\circledcirc}$, where $\circledcirc$ is a placeholder for $\uparrow$ or $\downarrow$, are called *hierarchical until and since*, and 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, **thr** $\mathcal{U}^{\uparrow}$ t_3 and **thr** $\mathcal{S}^{\downarrow}$ t_1 hold in position 2, because there is path 6-7-8 made of ending positions of chains starting in 2, such that **thr** holds until t_3 holds (or **thr** has held since t_1 held). Formulas **call** $\mathcal{U}^{\downarrow}$ p_c and **call** $\mathcal{S}^{\uparrow}$ p_b hold in position 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.

3.2. Formal 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 $i < j$ iff 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. Given $i, j \in U$, the chain relation 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. There exists at most one single position h s.t. $\chi(h, j)$ and $h \lessdot j$ or $h \doteq j$; for any k s.t. $\chi(k, j)$ and $k \gtrdot j$ we have $k > h$.
4. There exists at most one single position h s.t. $\chi(i, h)$ and $i \gtrdot h$ or $i \doteq h$; for any k s.t. $\chi(i, k)$ and $i \lessdot k$ we have $k < h$.

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 $\overrightarrow{\chi}(i, j) \iff \chi(i, j) \wedge (i \doteq j \vee i \gtrdot j)$ for any $i, j \in U$, and the *maximal backward* chain is defined as $\overleftarrow{\chi}(i, j) \iff \chi(i, j) \wedge (i \lessdot j \vee i \doteq j)$. In finite OP words, this implies $\overrightarrow{\chi}(i, j)$ iff $j = \max\{k \in U \mid \chi(i, k)\}$ and $\overleftarrow{\chi}(i, j)$ iff $i = \min\{k \in U \mid \chi(k, j)\}$. In Section 6, we shall see that this does not always hold in OP ω-words. The maximal forward (resp. backward) chain relation is undefined for a pair of positions either if 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 $\mathrm{a} \in AP$. Then, for any position $i \in U$ of w, we have $(w, i) \models \mathrm{a}$ if $\mathrm{a} \in P(i)$. 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 right (resp. left) context of a maximal chain that starts (resp. ends) in the current position: $(w, i) \models \bigcirc_\chi \varphi$ 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 Fig. 7, $(w, 3) \models \bigcirc_\chi \mathbf{thr}$ because $\overrightarrow{\chi}(3, 6)$ and $(w, 6) \models \mathbf{thr}$; $(w, 6) \models \ominus_\chi \mathbf{han}$ holds because $\overleftarrow{\chi}(2, 6)$ and $(w, 2) \models \mathbf{han}$, 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, $(w, i) \models \varphi\, \mathcal{U}^\Pi\, \psi$ 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 sets of paths.

The *linear until* ($\varphi\, \mathcal{U}\, \psi$) and *since* ($\varphi\, \mathcal{S}\, \psi$) 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 $\boxdot \subseteq \{\lessdot, \doteq, \gtrdot\}$, the $\mathcal{U}^\boxdot$ 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 $1 \leq k < n$,

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

There exists at most one forward OP-summary path between any two positions. For example, in Fig. 7, if we take $\boxdot = \{\gtrdot\}$ as in $(\mathbf{call} \vee \mathbf{thr})\, \mathcal{U}^{\gtrdot}\, \mathbf{ret}$, the path between 3 and 9 is made of positions 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. $\boxdot = \{\doteq, \lessdot\}$, there would be no such path, because consecutive positions in the $\gtrdot$ relation are not considered. With $\boxdot = \{\gtrdot, \lessdot\}$, 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 $\overrightarrow{\chi}$.

Definition 3.2. A *backward OP-summary* path is a sequence of positions $i = i_1 < i_2 < \cdots < i_n = j$ such that, for any $1 < k \leq n$,

$$i_{k-1} = \begin{cases} h & \text{if } \overleftarrow{\chi}(h, i_k) \text{ and } h \geq i; \\ i_k - 1 & \text{if } i_k - 1 \odot i_k \text{ with } \odot \in \boxdot, \text{ otherwise.} \end{cases}$$

For example, $(\mathbf{thr} \vee \mathbf{han})\, \mathcal{S}^{\lessdot}\, \mathbf{call}$ holds in position 8 because of path 1-2-8, that skips the body of chain $\overleftarrow{\chi}(2, 8)$ and satisfies $\mathbf{thr} \vee \mathbf{han}$ 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 $\mathcal{U}^{\uparrow}$ and $\mathcal{S}^{\downarrow}$ respectively, are based on paths made of the ending positions of non-maximal chains starting in the current position $i \in U$. The former 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 one.

$\mathcal{S}^{\downarrow}$ is called a since operator despite being a future modality. We chose this naming because in formulas such as $\varphi\, \mathcal{S}^{\downarrow}\, \psi$, 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 position 2 $\mathbf{thr}\,\mathcal{U}^{\uparrow}\, \mathrm{t}_3$ holds because of path 6-7-8, since **thr** holds in 6-7 and t_3 in 8; instead, the path made only of position 6 and the one made of 6-7, in which t_3 does not hold, do not satisfy it. Similarly, $\mathbf{thr}\,\mathcal{S}^{\downarrow}\, \mathrm{t}_1$ is satisfied by path 6-7-8, but not by the path made of position 8 and the one made of positions 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 ($\mathcal{U}^{\downarrow}$ and $\mathcal{S}^{\uparrow}$) 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 $\mathcal{U}^{\downarrow}$ is an until operator despite being a past modality: again, the reason is that $\varphi\, \mathcal{U}^{\downarrow}\, \psi$ enforces ψ at the end of the path, and φ in previous positions, making it more similar to an until operator. In position 6, $\mathbf{call}\,\mathcal{U}^{\downarrow}\, \mathrm{p}_c$ is satisfied by path 3-4 and not by the one made only of position 3, because p_c does not hold in 3, but it does in 4, and **call** holds in 3. Formula $\mathbf{call}\,\mathcal{S}^{\uparrow}\, \mathrm{p}_b$ 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.

3.3. Examples

Many relevant properties can be expressed in OPTL. In the paper, we use the standard shortcuts of LTL, such as $\Box$ and $\Diamond$, extended naturally to OPTL operators. E.g. $\Diamond^{\uparrow}\psi := \top\, \mathcal{U}^{\uparrow}\, \psi$, and $\Box^{\uparrow}\psi := \neg\Diamond^{\uparrow}\neg\psi$ are defined such that, evaluated in position i, mean that ψ holds in, resp., at least one and all positions j with $\chi(i, j)$ and not $\overrightarrow{\chi}(i, j)$. $\Diamond\!\!\!-^{\uparrow}\psi := \top\, \mathcal{S}^{\uparrow}\, \psi$ and $\boxminus^{\uparrow}\psi := \neg\Diamond\!\!\!-^{\uparrow}\neg\psi$ are symmetric.

Total/partial correctness. $\bigcirc^{\mathbf{ret}}\psi := \bigcirc_{\chi}(\mathbf{ret} \wedge \psi) \vee \bigcirc(\mathbf{ret} \wedge \psi)$, evaluated in a **call**, states that it is closed by a **ret** in which ψ holds. Thus, formula $\Box[\mathbf{call} \implies \bigcirc^{\mathbf{ret}}\top]$ holds if all procedures terminate, while it is false if there is an uncaught exception. We can also express Hoare-style pre- and post-conditions [33], which are used in most classical verification techniques. $\Box[(\mathbf{call} \wedge \mathrm{p}_A \wedge \rho) \implies \bigcirc^{\mathbf{ret}}\theta]$ expresses *total correctness*, i.e. whenever pre-condition ρ holds when procedure p_A is called, the latter terminates normally, with post-condition θ holding. Instead, $\Box[(\mathbf{call} \wedge \mathrm{p}_A \wedge \rho \wedge \bigcirc^{\mathbf{ret}}\top) \implies \bigcirc^{\mathbf{ret}}\theta]$ expresses *partial correctness*, i.e. the post-condition has to hold only when the procedure terminates normally.

Exception safety. We can do the same with **thr** statements: $\bigcirc^{\mathbf{thr}}\psi := \bigcirc_{\chi}(\mathbf{thr} \wedge \psi) \vee \bigcirc(\mathbf{thr} \wedge \psi)$, evaluated in a **call**, states that it is terminated by a **thr** in which ψ holds. $\Box[(\mathbf{call} \wedge \mathrm{p}_A) \implies \neg\bigcirc^{\mathbf{thr}}\top]$ is the requirement that procedure p_A never throws an exception, also known as the *no-throw guarantee*. When partial correctness is generalized to exceptions, we get *exception safety* [34], an important concept for the correctness of, especially, C++ programs. *Weak* exception safety requires that, when a C++ class member function terminates exceptionally, all class invariants are preserved (so the class instance is still in a functional state), and no resources are leaked. *Strong* exception safety adds the requirement that, in case of exceptional exit, the operation is aborted, and the state of the instance remains the same as it was before the member function was called. If θ is a class invariant, formula $\Box[(\mathbf{call} \wedge \mathrm{p}_A \wedge \theta \wedge \bigcirc^{\mathbf{thr}}\top) \implies \bigcirc^{\mathbf{thr}}\theta]$ expresses weak exception safety for p_A, and strong exception safety if θ represents the whole state of the class instance.

It is also possible to pose constraints on the **thr** statements caught by a handler: $\Diamond^{\uparrow}\psi$ holds in a handler if ψ holds in at least one of the **thr**s it catches. So, $\Box[\mathbf{han} \implies \neg(\Diamond^{\uparrow}(\mathbf{thr} \wedge \Diamond\!\!\!-^{\uparrow}\mathrm{p}_A) \wedge \Diamond^{\uparrow}(\mathbf{thr} \wedge \Diamond\!\!\!-^{\uparrow}\mathrm{p}_B))]$ means that a single handler cannot catch one or more exceptions that terminate both procedures p_A and p_B.

Function-local requirements. With OPM $M_{\mathbf{call}}$, the **call** of a procedure is the left context of non-maximal chains whose right contexts are the **call**s and **han**s it issues. Thus, $\Diamond^{loc}\psi := \Diamond^{\uparrow}((\mathbf{call} \wedge \psi) \vee (\mathbf{han} \wedge \Box^{\uparrow}(\mathbf{call} \implies \psi)))$ is true if ψ holds in one of the **call**s issued by the procedure represented by the **call** in which it is evaluated (even if they are guarded by a **han**). A "globally" version of this operator can be defined symmetrically. If we mark with wr_X the fact that a function writes the program variable X, then $\Box[\mathbf{call} \implies (\Diamond^{loc}\mathrm{wr}_X \implies \bigcirc^{\mathbf{thr}}\top)]$ requires that any function whose sub-calls write to X is terminated by an exception.

We did not include "internal" positions in OPM $M_{\mathbf{call}}$ for conciseness of the examples, but they could be defined with an $\doteq$-circularity as in OPM M^{NW} from Section 4, in order to represent program instructions that do not concern function invocation or termination. Then, function local requirements on such positions could be easily expressed with a $\mathcal{U}^{\doteq}$ operator.

Stack inspection. *Stack Inspection* gathers a wide range of requirements concerning the sequence of function frames that are present on the program's stack at a certain point of the execution. It is used to enforce security policies in, e.g., Java programs [19,35]. With the shortcut $\varphi\, \mathcal{S}^{\mathbf{call}}\, \psi := (\mathbf{call} \implies \varphi)\, \mathcal{S}^{\lessdot\doteq}\, (\mathbf{call} \wedge \psi)$ we get a since operator that only considers **call**s of procedures whose instances are active when the statement at the current word position is executed. This operator

is similar to the *call since* of CaRet [11], but it can also work in the presence of exceptions. We also add the related back operator $\ominus^{\mathbf{call}}\psi := (\neg\mathbf{call})\,\mathcal{S}^{\lessdot\doteq}\,(\mathbf{call}\wedge\psi)$, which enforces ψ in the **call** of the function on top of the stack at the point in the execution represented by a word position. Due to the separation property of LTL [36], these operators can express all first-order properties on the stack trace, subsuming the formalism of [35]. A typical requirement of this class is the following: function p_A can only be called by functions with privilege level l_1, an not by those with the lower privilege l_2. We can check this with formula $\Box[(\mathbf{call}\wedge\mathrm{p}_A)\implies(\neg\mathrm{l}_2)\,\mathcal{S}^{\mathbf{call}}\,\mathrm{l}_1]$. The latter is also expressible in CaRet, but in OPTL we may additionally state that, if this requirement is violated, an exception is thrown: $\Box[(\mathbf{call}\wedge\mathrm{p}_A\wedge((\neg\mathrm{l}_1)\,\mathcal{S}^{\mathbf{call}}\,\mathrm{l}_2))\implies\bigcirc^{\mathbf{thr}}\top]$. With a similar operator, formula $\Box[\mathbf{call}\wedge\mathrm{p}_b\implies\mathbf{call}\,\mathcal{S}^{\lessdot\doteq}\,\mathbf{han}]$ states that all calls to procedure p_b must be guarded by a **han** statement catching the exceptions they may throw.

With hierarchical operators, we can formulate requirements limited to the procedures in the stack trace that have been terminated by a single **thr** statement. Shortcut $\diamondsuit^{\uparrow}\psi$ is true in a **thr** position if ψ holds in at least one of the functions it terminates. So, we may express restrictions on which procedures may raise exceptions. E.g., $\Box[\mathbf{thr}\implies\neg\diamondsuit^{\uparrow}\mathrm{p}_b]$ means that procedure p_b cannot be terminated by an exception. $\Box[\mathbf{thr}\implies(\neg\mathrm{l}_2)\,\mathcal{S}^{\uparrow}\,\mathrm{l}_1]$ means that only functions with privilege level l_1 may throw, and those with level $\neg\mathrm{l}_2$ may only do so through an invocation to one of them.

Remark. Those of the requirements listed above that deal with exceptions are not expressible in NWTL, because it's nesting relation is one-to-one, and fails to model situations in which a single entity is in relation with multiple other entities. As we noted in Section 2, VPL can be used to model function calls and returns, which are in a one-to-one relation, but they cannot model the example of Fig. 7, because they have no way to express the many-to-one relation that holds between multiple function calls terminated by an exception, and the exception itself.

4. Relationship with nested words

We now explore the relationship between OPTL and the NWTL logic presented in [12]. NWTL is a temporal logic based on the VPL family, which is strictly contained in OPL. In [26,27,29] 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.

A nested word is a structure $NW = \langle U, (P_{\mathrm{a}})_{\mathrm{a}\in\Lambda}, <, \mu, \mathtt{call}, \mathtt{ret}\rangle$, where Λ is the set of atomic propositions; U is a set of word positions such that $U = \{1,\dots,n\}$ 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 two disjoint unary relations such that:

- $\mu(i,j)$ implies $\mathtt{call}(i)$ and $\mathtt{ret}(j)$ (a position cannot be both a `call` and a `ret`);
- if $\mu(i,j)$ and $\mu(i,j')$ hold then $j = j'$, while if $\mu(i,j)$ and $\mu(i',j)$ then $i = i'$ (i.e., μ is a *one-to-one* relation);
- and if $i \le j$ and $\mathtt{call}(i)$ and $\mathtt{ret}(j)$ then there exists a position k such that $i \le k \le j$, and either $\mu(i,k)$ or $\mu(k,j)$ (so the μ relation is not crossing, e.g. $\mu(1,3)$ and $\mu(2,4)$ cannot both hold, and unmatched `calls` and `rets` cannot be inside a pair of matched ones).

If $\mu(i,j)$ then i is the matching call of j, which is the matching return of i. If $\mathtt{call}(i)$ (resp. $\mathtt{ret}(j)$) 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).

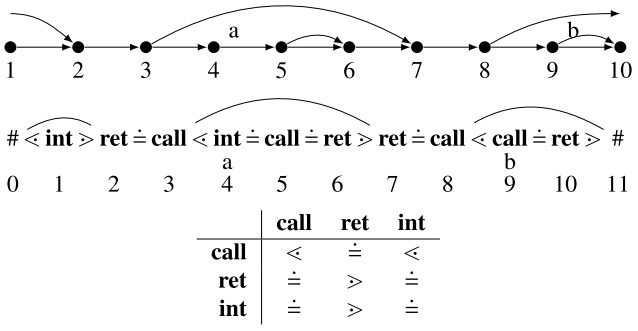
The syntax of NWTL is given by $\varphi := \top \mid \mathrm{a} \mid \mathtt{call} \mid \mathtt{ret} \mid \neg\varphi \mid \varphi\vee\varphi \mid \bigcirc\varphi \mid \bigcirc_\mu\varphi \mid \ominus\varphi \mid \ominus_\mu\varphi \mid \varphi\,\mathcal{U}^\sigma\,\varphi \mid \varphi\,\mathcal{S}^\sigma\,\varphi$, with $\mathrm{a}\in\Lambda$. The semantics of propositional and LTL-like operators is the usual one. For any $i\in U$, $(NW,i)\models\mathtt{call}$ iff $\mathtt{call}(i)$ and similarly for `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 $\mathcal{U}^\sigma$ and $\mathcal{S}^\sigma$ 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 $U = \{1,\dots,n\}$, we have $U' = U\cup\{0,n+1\}$. The set of propositional letters is $AP = \Lambda\cup\Sigma$ with $\Sigma = \{\mathbf{call},\mathbf{ret},\mathbf{int}\}$. 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 $\mathtt{call}(i)$, $\sigma(i) = \{\mathbf{ret}\}$ iff $\mathtt{ret}(i)$, and $\sigma(i) = \{\mathbf{int}\}$ otherwise. Finally, the OPM M^{NW} is shown in Fig. 8. Note that M^{NW} is different from the OPM used in [26] to prove that VPL are contained in OPL.

An example nested word is shown in Fig. 8, along with its translation into an OP 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 reasoning is formalized by the following lemmas.

Lemma 4.1. *For any two distinct positions $i, j\in U$, $i<j$, if $\chi(i,j)$ holds then* **call** $\in P'(i)$ *and* **ret** $\in P'(j)$.

Proof. $\chi(i,j)$ means i and j are the context of a chain. According to Definition 2.2, position i must yield precedence to the next position, $i+1$: $i\lessdot i+1$. According to matrix M^{NW}, only call positions can yield precedence to any other position



	call	ret	int
call	⋖	≐	⋖
ret	≐	⋗	≐
int	≐	⋗	≐

Fig. 8. The top 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} at the bottom.

(unless $\# \in P'(i)$, which is not the case), therefore $\mathbf{call} \in P'(i)$. Similarly, any position can take precedence from a return position only, and since $j-1$ must take precedence from j, we have $\mathbf{ret} \in P'(j)$. □

In Lemma 4.2 we prove that relation χ in OW is one-to-one if restricted to U. Note that this is not true for positions 0 and $n+1$: we have $\chi(0, j)$ for all $j \in U$ that are unmatched returns, and $\chi(i, n+1)$ for all unmatched calls $i \in U$.

Lemma 4.2. *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 $i = i'$.*

Proof. Suppose there exists a position $i \in U$ such that multiple chains start in i and end in distinct positions in U, i.e. there exist positions $j_1, \ldots, j_{n-1}, j_n \in U$ such that $i < j_1 < \cdots < j_{n-1} < j_n$ and $\chi(i, j_k)$ for any $1 \le k \le n$. Then, consider the outermost chain, $\chi(i, j_n)$: it must be a composed chain, because it contains the chain $\chi(i, j_{n-1})$. It must be of the form ${}^{a_i}[w_i a_{j_{n-1}} \ldots]^{a_{j_n}}$, hence we have $a_i \lessdot a_{j_{n-1}}$, and w_i is the body of chain ${}^{a_i}[w_i]^{a_{j_{n-1}}}$. But since $\chi(i, j_{n-1})$, and because of Lemma 4.1 $\mathbf{call} \in P'(i)$ and $\mathbf{ret} \in P'(j_{n-1})$. Therefore, according to M^{NW}, we have $a_i \doteq a_{j_{n-1}}$, which contradicts our previous claim. The fact that there exists no position $j \in U$ in which multiple chains starting in U end can be proved similarly. □

Lemma 4.3 shows that, when translating a nested word into an OP word, the μ relation of the former is reflected into the χ relation that holds in the latter. This is, as we observed previously, not true for consecutive positions.

Lemma 4.3. *For any $i, j \in U$ such that $j > i+1$, we have $\mu(i, j)$ iff $\chi(i, j)$.*

Proof. The proof is carried out by induction on the nesting depth of the μ/χ relation. Suppose $i, j \in U$ and $\mu(i, j)$. If all positions between i and j are internal (i.e. the nesting depth is 0), then $\mathbf{call} \in P'(i)$, $\mathbf{ret} \in P'(j)$ and $\mathbf{int} \in P'(k)$ for any $i < k < j$. The resulting word structure will be $i \lessdot i+1 \doteq \cdots \doteq j-1 \gtrdot j$, and $\chi(i, j)$. Now, suppose $\chi(i, j)$ is a simple chain. According to M^{NW}, if $i \lessdot i+1$ then $\mathbf{call} \in P'(i)$ and $\mathbf{int} \in P'(i+1)$, and similarly $\mathbf{int} \in P'(k)$ with $i < k < j$ and $\mathbf{ret} \in P'(j)$: in NW i is a call and j its matched return, so we have $\chi(i, j) \implies \mu(i, j)$.

Suppose $\mu(i, j)$, and there are other call and return positions between i and j. By the definition of μ, such positions are balanced, i.e., each `call` between i and j has a matching `ret` before j. So, the nested word between i and j has the form $c_0 a_1 x_1 \ldots x_{n-1} a_{n-1} r_n$ where $c_0 = i$, $r_n = j$, $\texttt{call}(c_0)$, $\texttt{ret}(r_n)$, and for any p such that $0 < p < n$, $a_p = \varepsilon$ or it is an internal position, and either $x_p = \varepsilon$ or x_p is a nested word of the form $x_p = c_p y_p r_p$, with $\texttt{call}(c_p)$ and $\texttt{ret}(r_p)$. Once translated into an OP word, this results in the composed chain ${}^{c_0}[a_1 x_1 \ldots x_{n-1} a_{n-1}]^{r_n}$, so $\chi(i, j)$. Indeed, according to our translation, $\mathbf{call} \in P'(i)$ and, if $a_1 \neq \varepsilon$, then $a_1 = i+1$ and $\mathbf{int} \in P'(i+1)$, which implies $c_0 \lessdot a_1$; if $a_1 = \varepsilon$ then i is followed by $c_1 = i+1$ (which is part of x_1) and $\mathbf{call} \in P'(i+1)$, so $c_0 \lessdot c_1$. Similarly, either $a_{n-1} \gtrdot r_n$ or $r_{n-1} \gtrdot r_n$. Also, each x_p forms a chain ${}^{c_p}[y_p]^{r_p}$ by itself, while ${}^{c_0}[a_1 c_1 r_1 \ldots c_{n-1} r_{n-1} a_{n-1}]^{r_n}$ is a simple chain.

The fact that if $\chi(i, j)$ is a composed chain, then $\mu(i, j)$ can be proved analogously, keeping into account the results of Lemmas 4.1 and 4.2. □

The following lemma establishes a correspondence between summary paths in NWTL and OP Summary paths in OPTL, enabling the translation of NWTL summary until operators with their operator-precedence counterparts.

Lemma 4.4. *Given any two word positions $i, j \in U$, $i \le j$, the summary path between i and j in NW coincides with the Operator Precedence summary path between the same positions based on precedence relations $\boxdot = \{\lessdot, \doteq, \gtrdot\}$ in OW.*

Proof. As reported in [12], a summary path between $i, j \in U$, $i < j$, is a sequence $i = i_1 < i_2 < \cdots < i_k = j$ such that for all $p < k$

$$i_{p+1} = \begin{cases} r(i_p) & \text{if } i_p \text{ is a matched call and } j \geq r(i_p); \quad \text{(a)} \\ i_p + 1 & \text{otherwise;} \quad \text{(b)} \end{cases} \tag{2}$$

where $r(i_p)$ is the only position such that $\mu(i_p, r(i_p))$, if it exists.

The definition of Operator Precedence Summary Paths is given in Section 3.1. Note that, because of Lemma 4.2, we have $\chi(i,j) \iff \overleftarrow{\chi}(i,j) \iff \overrightarrow{\chi}(i,j)$ for any $i, j \in U$, and OP Forward and Backward Summary paths coincide. Moreover, due to Lemma 4.3, $\chi(i,j) \iff \mu(i,j)$ if $j > i+1$: case (2a) of the definition above coincides with the first case of Definition 3.1. If $j = i+1$ and $\mu(i,j)$, $\chi(i,j)$ does not hold, but this case is subsumed by the second case of Definition 3.1. The latter also incorporates case (2b) of the definition of summary path in NWTL, because set $\boxdot$ includes all possible precedence relations. Since we have proved that the two definitions coincide for each single step of the path, it is possible to inductively prove that the two kinds of paths actually coincide. □

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 4.5 *(NWTL ⊆ 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 structure 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 define $\varphi' = \alpha(\varphi)$ inductively as follows, for non-trivial operators:

- $\alpha(\bigcirc_\mu \varphi) = \bigcirc_\chi \alpha(\varphi) \vee (\mathbf{call} \wedge \bigcirc(\mathbf{ret} \wedge \alpha(\varphi)))$. The validity of this translation trivially follows from Lemma 4.3. Unfortunately, the double repetition of $\alpha(\varphi)$ may cause an exponential blowup in the worst-case length of the translation.
  The following more complex translation does not suffer from this issue: $\alpha(\bigcirc_\mu \varphi) = \mathbf{call} \wedge \mathbf{call}\, \mathcal{U}^{\doteq}\, (\mathbf{ret} \wedge \alpha(\varphi))$. To better explain it, let $\gamma := \mathbf{call}\, \mathcal{U}^{\doteq}\, (\mathbf{ret} \wedge \alpha(\varphi))$. In NWTL we have $(w,i) \models \bigcirc_\mu \varphi$ iff there exists $j \in U$ such that $\mu(i,j)$ and $(w,j) \models \varphi$. When referring to the OP structure, we must distinguish between a few mutually exclusive cases:
  - *A position j such that $\mu(i,j)$ exists and $j > i+1$.* Then, by Lemma 4.3 also $\chi(i,j)$ holds and, because of Lemma 4.2, 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. Furthermore, this is the only path in which γ is true; in fact, paths terminating in a position strictly between i and j are forbidden by allowing only the $\doteq$ relation, and all paths surpassing j must include it, but $\mathbf{call} \notin P'(j)$ falsifies γ.
  - *A position j such that $\mu(i,j)$ exists, but $j = i+1$.* In this case, 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.
  - *i is a pending call.* Then $\chi(i, n+1)$, but $\mathbf{ret} \notin P'(n+1)$, which falsifies γ.
  - *i is an internal position.* $(w,i) \not\models \bigcirc_\mu \varphi$ because position i is not a matched call, so $\mathbf{int} \in P'(i)$ and γ is false because $\mathbf{call}, \mathbf{ret} \notin P'(i)$.
  - *i is a return.* If $\mathtt{ret}(i)$, then $\mathbf{call}$ in $\alpha(\bigcirc_\mu \varphi)$ is false in i.
- $\alpha(\ominus_\mu \varphi) = \mathbf{ret} \wedge \mathbf{ret}\, \mathcal{S}^{\doteq}\, (\mathbf{call} \wedge \alpha(\varphi))$. The argument that justifies this equivalence is similar to the previous one. Again, the more straightforward translation $\alpha(\ominus_\mu \varphi) = \ominus_\chi \alpha(\varphi) \vee (\mathbf{ret} \wedge \ominus(\mathbf{call} \wedge \alpha(\varphi)))$ causes an exponential blowup in formula length.
- $\alpha(\varphi\, \mathcal{U}^\sigma\, \psi) = \alpha(\varphi)\, \mathcal{U}^{\gtrdot \doteq \lessdot}\, \alpha(\psi)$: from Lemma 4.4 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.
- $\alpha(\varphi\, \mathcal{S}^\sigma\, \psi) = \alpha(\varphi)\, \mathcal{S}^{\gtrdot \doteq \lessdot}\, \alpha(\psi)$: the justification is analogous to the until case.

By induction on the syntactic structure of φ, we can conclude that $(w,i) \models \varphi$ iff $(w',i) \models \varphi'$, and consequently NWTL ⊆ OPTL. □

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

The above translation can easily be extended to ω-words, which are treated in Section 6.

It is easy to see that the OPTL semantics of Section 3.2 is expressible in First-Order (FO) Logic, so OPTL ⊆ FO. Thus, by the FO-completeness result for NWTL of [12], we conclude

Corollary 4.6. *If OPTL is restricted to OPM M^{NW}, we have OPTL = NWTL = FO.*

This does not mean that OPTL has the same expressive power of NWTL. In fact, the greater expressiveness of OPTL derives from that of OPL w.r.t. VPL, and it consists in using more general OPMs, such as $M_{\mathbf{call}}$. In general, OPTL formulae exploiting

the fact that the χ relation is not exclusively one-to-one, such as those pointed out in Section 3.3, express properties not expressible in NWTL. Thus, we can state the following:

Corollary 4.7. *NWTL* $\subset$ *OPTL.*

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 a nondeterministic OPA $\mathcal{A}_\varphi$ accepting the finite strings on $\mathscr{P}(AP)$ satisfying φ, compatible with $M_{\mathscr{P}(AP)}$. The way to extend this definition to infinite words is nontrivial, and we detail it in Section 6.

Let $\mathcal{A}_\varphi = \langle \mathscr{P}(AP), M_{\mathscr{P}(AP)}, \text{Atoms}(\varphi), I, F, \delta \rangle$ be an OPA, with $M_{\mathscr{P}(AP)}$ being an OPM, and $\text{Atoms}(\varphi)$ being the set of states. The construction of this OPA follows the classical lines for LTL and the subsequent ones for NWTL, but it requires many more nontrivial technicalities, due to the more general structure provided by the OPM w.r.t. the linear and the VPL structures. A set called *closure* and denoted by $\text{Cl}(\varphi)$ is defined, and it contains all possible subformulas of φ, plus some additional auxiliary operators (in set $\text{Cl}_{aux}(\varphi)$), which will be introduced later in this section. The set $\text{Atoms}(\varphi)$ only contains logically consistent subsets of $\text{Cl}(\varphi)$. The notion of consistency depends on the semantics of each operator, and we define it separately for each one of them in the following sections, together with constraints on the transition relation δ, which enforce the temporal requirements.

The correctness, i.e., soundness and completeness, proof of this procedure is given in Section 5.9. It is carried out by induction on the syntactic structure of φ, so we prove the correctness of the construction for each operator separately, together with the related construction rules.

5.1. LTL operators

We first describe how the OPA is built for LTL-like operators, which is done with the classic procedure of [37], and then treat other operators separately. Initially, the *closure* $\text{Cl}(\varphi)$ of a formula φ is defined as the smallest set such that

1. $\varphi \in \text{Cl}(\varphi)$,
2. $AP \subseteq \text{Cl}(\varphi)$,
3. if $\psi \in \text{Cl}(\varphi)$ and $\psi \neq \neg\theta$ for any OPTL formula θ, then $\neg\psi \in \text{Cl}(\varphi)$;
4. if any of $\neg\psi$, $\bigcirc\psi$ or $\ominus\psi$ is in $\text{Cl}(\varphi)$, then $\psi \in \text{Cl}(\varphi)$;
5. if any of $\psi \wedge \theta$ or $\psi \vee \theta$ is in $\text{Cl}(\varphi)$, then $\psi \in \text{Cl}(\varphi)$ and $\theta \in \text{Cl}(\varphi)$.
6. if $\psi \,\mathcal{U}\, \theta \in \text{Cl}(\varphi)$, then $\psi, \theta, \bigcirc(\psi \,\mathcal{U}\, \theta) \in \text{Cl}(\varphi)$.

We will further enrich $\text{Cl}(\varphi)$ with auxiliary operators when dealing with non-LTL operators. We denote $\text{Cl}_{aux}(\varphi) \subset \text{Cl}(\varphi)$ the set of such operators.

The set $\text{Atoms}(\varphi)$ contains all consistent sets $\Phi \subseteq \text{Cl}(\varphi)$, i.e., such that

1. for every $\psi \in \text{Cl}(\varphi)$, $\psi \in \Phi$ iff $\neg\psi \notin \Phi$;
2. if $\psi \wedge \theta \in \Phi$, then $\psi \in \Phi$ and $\theta \in \Phi$;
3. if $\psi \vee \theta \in \Phi$, then $\psi \in \Phi$ or $\theta \in \Phi$;
4. if $\psi \,\mathcal{U}\, \theta \in \Phi$, then either $\theta \in \Phi$, or $\psi \in \Phi$ and $\bigcirc(\psi \,\mathcal{U}\, \theta) \in \Phi$; the rules for the linear since operator are analogous.

The set of initial states I consists only of atoms containing φ, but with no $\ominus$ or $\mathcal{S}$ formula. The set of final states F contains all atoms $\Phi \in \text{Atoms}(\varphi)$ not containing any $\bigcirc$ or $\mathcal{U}$ formula, so no word terminating with unsatisfied temporal requirements can be accepted, and such that $AP \cap \Phi = \{\#\}$, 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 $\Phi, \Theta \in \text{Atoms}(\varphi)$ and $a \in \mathscr{P}(AP)$, let $(\Phi, a, \Theta) \in \delta_{push} \cup \delta_{shift}$:

1. for any $\text{p} \in AP$, $\text{p} \in \Phi$ iff $\text{p} \in a$;
2. $\bigcirc\psi \in \Phi$ iff $\psi \in \Theta$;
3. $\ominus\psi \in \Theta$ 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 $\Phi, \Theta, \Psi \in \text{Atoms}(\varphi)$, $(\Phi, \Theta, \Psi) \in \delta_{pop}$ only if Ψ is the smallest set such that $\Phi \setminus \text{Cl}_{aux}(\varphi) \subseteq \Psi$, and for any $\text{p} \in AP$, $\text{p} \in \Phi$ iff $\text{p} \in \Psi$. Ψ may also contain additional formulas required by the rules detailed in the next sections.

5.2. 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. To do so, we exploit the following property of OPA:

Lemma 5.1. *Let M be an OPM, and w an OP word structure on it such that $\chi(i, j)$, for some positions $i, j \in w$. Let $\mathcal{A}$ be an OPA on M.*

- *If $\overrightarrow{\chi}(i, j)$, then $\mathcal{A}$ performs a pop or a shift transition after the last pop of the support of $\chi(i, j)$.*
- *If $\overleftarrow{\chi}(i, j)$, then $\mathcal{A}$ performs a push or a shift transition after the last pop of the support of $\chi(i, j)$.*

Proof. $\mathcal{A}$ reads position i with a push or a shift move, hence α, the topmost stack symbol before the support of $\chi(i, j)$ starts, contains the label of i. By Definition 2.5, the support of $\chi(i, j)$ ends with a pop move, with the label of j as the next input symbol. After such move, α is on top of the stack. If $\overrightarrow{\chi}(i, j)$, by definition of forward-maximal chain, either $i \gtrdot j$, and α is popped, or $i \doteq j$, and α is updated by the shift move reading j. If $\overleftarrow{\chi}(i, j)$, either $i \lessdot j$ or $i \doteq j$, so j is read by, resp., a push or a shift transition. □

In the following, by saying that a chain starts (resp. ends) in a position i, we mean i is its left (resp. right) context, i.e. $\chi(i, j)$ (resp. $\chi(j, i)$) for some j. In this construction, we introduce two of the auxiliary symbols in $\mathrm{Cl}_{aux}(\varphi)$ we mentioned before. The OPA 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 maximal chain begins, the automaton stores the auxiliary symbol $\bigcirc^s_\chi \psi$ onto the stack, and pops it when that chain ends, forcing ψ to hold in the appropriate state.

Formally, if $\bigcirc_\chi \psi \in \mathrm{Cl}(\varphi)$, then we add $\psi \in \mathrm{Cl}(\varphi)$ and $\bigcirc^s_\chi \psi, \bigcirc^{end}_\chi \psi \in \mathrm{Cl}_{aux}(\varphi)$. Also, we enrich the previous constraints on δ as follows: for any $\Phi, \Theta, \Psi \in \mathrm{Atoms}(\varphi)$ and $a \in \mathscr{P}(AP)$, let $(\Phi, a, \Theta) \in \delta_{push} \cup \delta_{shift}$, then

1. $\bigcirc_\chi \psi \in \Phi$ iff $\bigcirc^s_\chi \psi \in \Theta$;
2. if $\bigcirc_\chi \psi \in \Phi$, then $\bigcirc^{end}_\chi \psi \notin \Theta$;

let $(\Phi, a, \Theta) \in \delta_{shift}$ or $(\Phi, \Theta, \Psi) \in \delta_{pop}$, then

3. $\bigcirc^{end}_\chi \psi \in \Phi$ iff $\psi \in \Phi$;
4. $\bigcirc^s_\chi \psi \notin \Phi$;

let $(\Phi, \Theta, \Psi) \in \delta_{pop}$, then

5. $\bigcirc^s_\chi \psi \in \Theta$ iff $\bigcirc^s_\chi \psi \in \Psi$ or $\bigcirc^{end}_\chi \psi \in \Psi$.

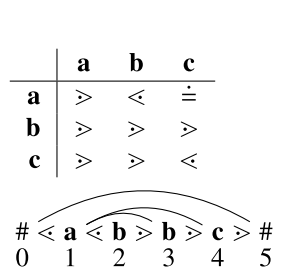
Finally, for any formula $\bigcirc_\chi \psi \in \mathrm{Cl}(\varphi)$, we must exclude from F all atoms containing $\bigcirc^s_\chi \psi$ or $\bigcirc_\chi \psi$.

Before formally proving the correctness of this construction, we explain how it works by means of the example run of the automaton for formula $\bigcirc_\chi \mathbf{c}$ of Fig. 9. In the following, structural labels are typeset in bold. $\bigcirc_\chi \mathbf{c}$ holds in the initial state: since the symbol # yields precedence to all other symbols, the first one (namely $\{\mathbf{a}\}$) is consumed by a push transition. Then, by rule 1, the OPA transitions to a state q_1 containing the auxiliary operator $\bigcirc^s_\chi \psi$. The latter is kept in either the current state or a stack symbol until the temporal requirement of $\bigcirc_\chi \mathbf{c}$ is satisfied. A chain starts in $\{\mathbf{a}\}$, so the next symbol $\{\mathbf{b}\}$ is also read by a push transition, which stores the previous state q_1, containing $\bigcirc^s_\chi \mathbf{c}$, on the stack (step 2-3 of Fig. 9c). Then, the chain ends with the second $\{\mathbf{b}\}$ symbol, and state q_1 is popped. $\mathbf{c}$, the operand of $\bigcirc_\chi$, must hold at the end of the maximal chain, and this is not maximal. However, the OPA still cannot discern between the two cases, and must wait for the next transition to occur.

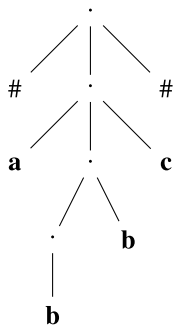
So, the OPA *nondeterministically guesses* that this is not a maximal chain by transitioning to a state (q_1) not containing $\bigcirc^{end}_\chi \psi$, which would mark the end of such a chain. By rule 5, $\bigcirc^s_\chi \mathbf{c}$ is instead present in q_1 (step 3-4). By Lemma 5.1, the automaton performs a shift or a pop move when it encounters the right context of a forward-maximal chain, and the topmost stack symbol contains its left context.

In step 4, the topmost stack symbol contains $\{\mathbf{a}\}$ (the left context of the chain), but $\{\mathbf{a}\} \lessdot \{\mathbf{b}\}$, so the next transition is a push, meaning $\{\mathbf{b}\}$ is the right context of a non-maximal chain. Thus, the current state q_1, containing $\bigcirc^s_\chi \mathbf{c}$, is again pushed on stack (step 4-5).

Between steps 5-6 the OPA *nondeterministically guesses* the end of the maximal chain, and by rule 5 $\bigcirc^{end}_\chi \psi \in q_4$. In step 6, the symbol containing $\{\mathbf{a}\}$ is, again, on top of the stack. Since $\{\mathbf{a}\} \doteq \{\mathbf{c}\}$, the next transition is a shift, meaning the chain between $\{\mathbf{a}\}$ and $\{\mathbf{c}\}$ is maximal. The transition relation is constrained by rule 3 so that this shift transition cannot occur unless also $\mathbf{c}$ is present in the same state, fulfilling the temporal requirement of $\bigcirc_\chi \mathbf{c}$. This is the case in Fig. 9c, so the computation goes on, accepting the input word.



(a)



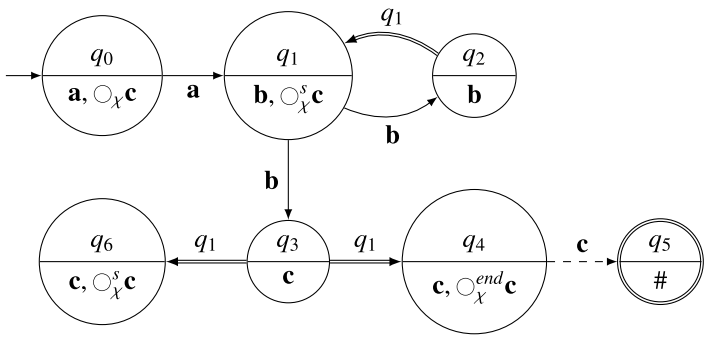
(b)

step	input	state	stack	relation	next move	rule
1	$\mathbf{a} \lessdot \mathbf{b} \gtrdot \mathbf{b} \gtrdot \mathbf{c} \gtrdot \#$	$q_0 = \{\mathbf{a}, \bigcirc_\chi \mathbf{c}\}$	$\bot$	$\# \lessdot \mathbf{a}$	push	1, 2
2	$\mathbf{b} \gtrdot \mathbf{b} \gtrdot \mathbf{c} \gtrdot \#$	$q_1 = \{\mathbf{b}, \bigcirc^s_\chi \mathbf{c}\}$	$[\mathbf{a}, q_0]\bot$	$\mathbf{a} \lessdot \mathbf{b}$	push	–
3	$\mathbf{b} \gtrdot \mathbf{c} \gtrdot \#$	$q_2 = \{\mathbf{b}\}$	$[\mathbf{b}, q_1][\mathbf{a}, q_0]\bot$	$\mathbf{b} \gtrdot \mathbf{b}$	pop	5
4	$\mathbf{b} \gtrdot \mathbf{c} \gtrdot \#$	$q_1 = \{\mathbf{b}, \bigcirc^s_\chi \mathbf{c}\}$	$[\mathbf{a}, q_0]\bot$	$\mathbf{a} \lessdot \mathbf{b}$	push	–
5	$\mathbf{c} \gtrdot \#$	$q_3 = \{\mathbf{c}\}$	$[\mathbf{b}, q_1][\mathbf{a}, q_0]\bot$	$\mathbf{b} \gtrdot \mathbf{c}$	pop	5
6	$\mathbf{c} \gtrdot \#$	$q_4 = \{\mathbf{c}, \bigcirc^{end}_\chi \mathbf{c}\}$	$[\mathbf{a}, q_0]\bot$	$\mathbf{a} \doteq \mathbf{c}$	shift	3, 4
7	#	$q_5 = \{\#\}$	$[\mathbf{c}, q_0]\bot$	$\mathbf{c} \gtrdot \#$	pop	–
8	#	q_5	$\bot$	$\# \doteq \#$	–	–

(c)

step	input	state	stack	relation	next move	rule
1	$\mathbf{a} \lessdot \mathbf{b} \gtrdot \mathbf{b} \gtrdot \mathbf{c} \gtrdot \#$	$q_0 = \{\mathbf{a}, \bigcirc_\chi \mathbf{c}\}$	$\bot$	$\# \lessdot \mathbf{a}$	push	1, 2
2	$\mathbf{b} \gtrdot \mathbf{b} \gtrdot \mathbf{c} \gtrdot \#$	$q_1 = \{\mathbf{b}, \bigcirc^s_\chi \mathbf{c}\}$	$[\mathbf{a}, q_0]\bot$	$\mathbf{a} \lessdot \mathbf{b}$	push	–
3	$\mathbf{b} \gtrdot \mathbf{c} \gtrdot \#$	$q_2 = \{\mathbf{b}\}$	$[\mathbf{b}, q_1][\mathbf{a}, q_0]\bot$	$\mathbf{b} \gtrdot \mathbf{b}$	pop	5
4	$\mathbf{b} \gtrdot \mathbf{c} \gtrdot \#$	$q_1 = \{\mathbf{b}, \bigcirc^s_\chi \mathbf{c}\}$	$[\mathbf{a}, q_0]\bot$	$\mathbf{a} \lessdot \mathbf{b}$	push	–
5	$\mathbf{c} \gtrdot \#$	$q_3 = \{\mathbf{c}\}$	$[\mathbf{b}, q_1][\mathbf{a}, q_0]\bot$	$\mathbf{b} \gtrdot \mathbf{c}$	pop	5
6	$\mathbf{c} \gtrdot \#$	$q_6 = \{\mathbf{c}, \bigcirc^s_\chi \mathbf{c}\}$	$[\mathbf{a}, q_0]\bot$	$\mathbf{a} \doteq \mathbf{c}$	shift	4

(d)



(e)

Fig. 9. An OPM and word **abbc** (9a), its AST (9b), an accepting run of the OPA for formula $\bigcirc_\chi \mathbf{c}$ on it (9c), a non-accepting one (9d), and the part of the OPA involved in such runs (9e). Note that the whole OPA has 64 states and a much denser transition relation. The OPA states are represented both explicitly as atoms and with a numeric identifier (e.g. q_i), for clarity. We write **a** instead of $\{\mathbf{a}\}$ for singletons containing one structural label.

Instead, runs that took the wrong guess, such as the one of Fig. 9d, cannot continue. By rule 4, the shift transition relation is only defined if the starting state does not contain $\bigcirc^s_\chi \mathbf{c}$. Since $\bigcirc^s_\chi \mathbf{c} \in q_6$, the relation is undefined, and this run stops.

If we had $\{\mathbf{a}\} \gtrdot \{\mathbf{c}\}$, a pop move would take place instead of the final shift, and rules 3 and 4 would still hold. Fig. 10 shows an accepting run on such a word.

Notice how the OPA exploits its own semantics to discern between maximal and non-maximal chains: by Lemma 5.1, a pop followed by a push corresponds to the end of a non-maximal chain, and a pop followed by a pop or a shift to the end of a maximal chain. Which kind of transition is performed only depends on precedence relations between input symbols, and not on the current state of the automaton. Thus, it suffices to build the transition relation so that, when the pop or shift

step	input	state	stack	relation	next move	rule
1	$\mathbf{a} \lessdot \mathbf{b} \gtrdot \mathbf{b} \gtrdot \mathbf{c} \gtrdot \#$	$q_0 = \{\mathbf{a}, \bigcirc_\chi \mathbf{c}\}$	$\bot$	$\# \lessdot \mathbf{a}$	push	1, 2
2	$\mathbf{b} \gtrdot \mathbf{b} \gtrdot \mathbf{c} \gtrdot \#$	$q_1 = \{\mathbf{b}, \bigcirc^s_\chi \mathbf{c}\}$	$[\mathbf{a}, q_0]\bot$	$\mathbf{a} \lessdot \mathbf{b}$	push	–
3	$\mathbf{b} \gtrdot \mathbf{c} \gtrdot \#$	$q_2 = \{\mathbf{b}\}$	$[\mathbf{b}, q_1][\mathbf{a}, q_0]\bot$	$\mathbf{b} \gtrdot \mathbf{b}$	pop	5
4	$\mathbf{b} \gtrdot \mathbf{c} \gtrdot \#$	$q_1 = \{\mathbf{b}, \bigcirc^s_\chi \mathbf{c}\}$	$[\mathbf{a}, q_0]\bot$	$\mathbf{a} \lessdot \mathbf{b}$	push	–
5	$\mathbf{c} \gtrdot \#$	$q_3 = \{\mathbf{c}\}$	$[\mathbf{b}, q_1][\mathbf{a}, q_0]\bot$	$\mathbf{b} \gtrdot \mathbf{c}$	pop	5
6	$\mathbf{c} \gtrdot \#$	$q_4 = \{\mathbf{c}, \bigcirc^{end}_\chi \mathbf{c}\}$	$[\mathbf{a}, q_0]\bot$	$\mathbf{a} \gtrdot \mathbf{c}$	pop	3, 4
7	$\mathbf{c} \gtrdot \#$	q_3	$\bot$	$\# \lessdot \mathbf{c}$	push	–
8	$\#$	$q_5 = \{\#\}$	$[\mathbf{c}, q_3]\bot$	$\mathbf{c} \gtrdot \#$	pop	–
9	$\#$	q_5	$\bot$	$\# \doteq \#$	–	–

Fig. 10. An accepting run on the OP word of Fig. 9, but with an OPM that only differs because $\{\mathbf{a}\} \gtrdot \{\mathbf{c}\}$.

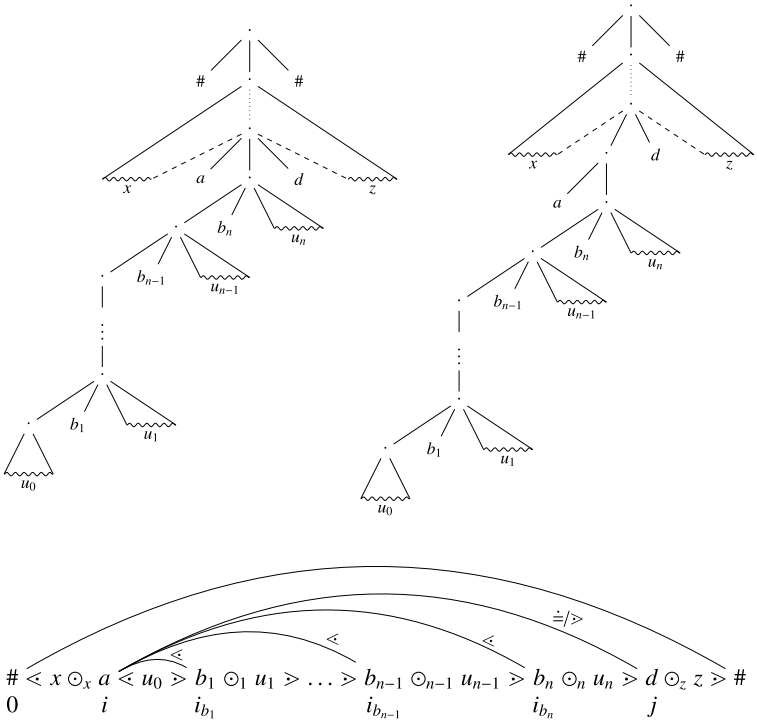


Fig. 11. The two possible ASTs of a generic OP word $w = xyz$ (top), and its flat representation with chains (bottom). Wavy lines are placeholders for subtree frontiers. We have either $a \doteq d$ (top left), or $a \gtrdot d$ (top right). In both trees, $a \lessdot b_k$ for $1 \leq k \leq n$, and the corresponding word positions are in the chain relation. For $1 \leq k \leq n$, u_k is the word generated by the right part of the rhs whose first terminal is b_k. So, either ${}^{b_k}[u_k]^{b_{k+1}}$, or u_k is of the form $v^k_0 c^k_0 v^k_1 c^k_1 \ldots c^k_{m_k} v^k_{m_k+1}$, where $c^k_p \doteq c^k_{p+1}$ for $0 \leq p < m_k$, $b_k \doteq c^k_0$, and resp. $c^k_{m_k} \gtrdot b_{k+1}$ and $c^n_{m_n} \gtrdot d$ (cf. Fig. 12). Moreover, for each $0 \leq p < m_k$, either $v^k_{p+1} = \varepsilon$ or ${}^{c^k_p}[v^k_{p+1}]^{c^k_{p+1}}$; either $v^k_0 = \varepsilon$ or ${}^{b_k}[v^k_0]^{c^k_0}$, and either $v^k_{m_k+1} = \varepsilon$ or ${}^{c^k_{m_k}}[v^k_{m_k+1}]^{b_{k+1}}$ (resp. ${}^{c^n_{m_n}}[v^n_{m_n+1}]^d$). u_0 has this latter form, except $v^0_0 = \varepsilon$ and $a \lessdot c^0_0$. In the bottom representation, $\odot$s are placeholders for precedence relations, that depend on the surrounding characters.

move revealing the end of a maximal chain occurs, only runs satisfying the argument of $\bigcirc_\chi$ in that position may continue (rules 3 and 4). All other rules are needed to keep track of such temporal requirements until they are satisfied.

The correctness of the construction is proved in Lemma 5.2. In its proof, we use Fig. 11, which represents as ASTs the two possible structures that a generic OP word $w = xyz$ with a forward-maximal chain may have, depending on the precedence relations. In it, dots are non-terminal symbols, and each one of them corresponds to a chain, whose contexts are the surrounding terminals. a and d[3] are the contexts of a maximal chain, and in the left AST we have $a \doteq d$, and $a \gtrdot d$ in the right one. y is the sub-word between a and d, and it represents a maximal chain with a generic body. The leftmost branch of the AST of such body is expanded, to show all non-maximal chains whose left context is a. Indeed, the chain relation holds between a and all terminals b_k, $1 \leq k \leq n$, and $a \lessdot b_k$. Sub-words u_k, are the remaining parts of the rhs starting with b_k, and they can be of any form: they may be empty, or the body of a chain between b_k and b_{k+1}, or a more generic rhs, as shown in Fig. 12. u_0 is the body of the innermost chain whose left context is a. In order to understand the proof of

[3] Recall that italic symbols represent subsets of AP, whose precedence relations are determined by the structural labels they contain.

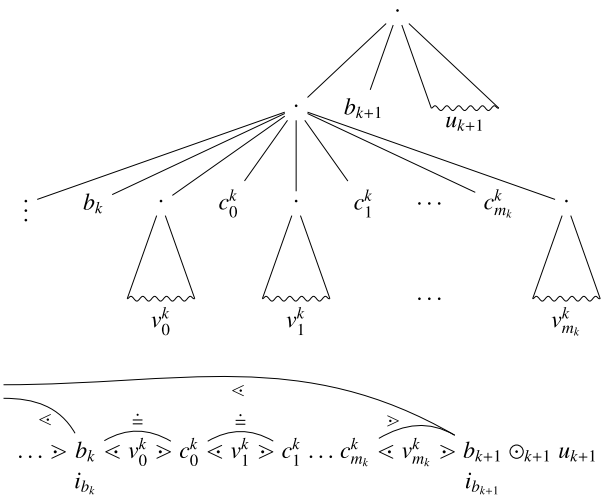


Fig. 12. The structure of u_k in the word of Fig. 11.

Lemma 5.2, the reader should familiarize with the way an OPA would read such a word. In general, a could be read by a shift or a push move. Then, the first terminal of u_0 is read by a push move, and the pushed stack symbol is popped only when b_1 is reached, since u_0 is read by only pushing and popping other stack symbols. b_1 is then read by another push, and the corresponding push symbol is popped when reaching b_2, and so on. When d is reached, the stack symbol corresponding to b_n is popped and, if $a \gtrdot d$, also the symbol pushed or updated when reading a is popped, while if $a \doteq d$, such symbol is just updated by a shift move.

Lemma 5.2. *Let AP be a finite set of atomic propositions, $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$ an OP alphabet, ψ an OPTL formula on it, and $\mathcal{A}_{\bigcirc_\chi \psi}$ an OPA satisfying rules 1-5 of Section 5.2. For any word $w = \#xyz\#$ on $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$, let position $i = |x| + 1$ in w: we have*

$$(w, i) \models \bigcirc_\chi \psi$$

if and only if $\mathcal{A}_{\bigcirc_\chi \psi}$ performs one computation that brings it from configuration $C_0 = \langle yz, \Phi, \alpha\gamma\rangle$ with $\bigcirc_\chi \psi \in \Phi$ to $\langle z, \Phi', \alpha'\gamma\rangle$, such that $\bigcirc^s_\chi \psi, \bigcirc^{end}_\chi \psi \notin \Phi'$, $|\alpha| = 1$ and $|\alpha'| \leq 1$.

Moreover, if $(w, i) \models \bigcirc_\chi \psi$, all accepting computations of $\mathcal{A}_{\bigcirc_\chi \psi}$ pass through C_0.

Proof. [$\Rightarrow$] If $\bigcirc_\chi \psi$ holds in i, then w is in the form of Fig. 11: at least one chain starts in i, and ψ holds in the position j such that $\overrightarrow{\chi}(i, j)$.

In the following, we denote as Φ_a (resp. Φ_x) the state of the automaton right before reading symbol $a \in \mathscr{P}(AP)$ (resp. $x \in \mathscr{P}(AP)^+$). Suppose the OPA is in configuration $C_0 = \langle a \ldots z, \Phi_a, [f, \Phi_f]\gamma\rangle$, where $a = \Phi_a \cap AP$, $\alpha = [f, \Phi_f]$. The OPA guesses that $\bigcirc_\chi \psi$ holds in the current position, so $\bigcirc_\chi \psi \in \Phi_a$. Symbol a is read by either a push or a shift transition, depending on the precedence relation between f and it. The new configuration is $\langle c_0^0 \ldots z, \Phi_{c_0^0}, \sigma\rangle$, where $\sigma = [a, \Phi_f]\gamma$ if $f \doteq a$, and $\sigma = [a, \Phi_a][f, \Phi_f]\gamma$ if $f \lessdot a$. Due to rule 1, we have $\bigcirc^s_\chi \psi \in \Phi_{c_0^0}$. Now, since $a \lessdot c_0^0$, the next move is a push, and the resulting configuration is $\langle v_1^0 \ldots z, \Phi_{v_1^0}, [c_0^0, \Phi_{c_0^0}]\sigma\rangle$. Then, if $v_1^0 \neq \varepsilon$, the automaton reads it. Since ${}^{c_0^0}[v_1^0]^{c_1^0}$ is a chain, it eventually comes to a configuration $\langle c_1^0 \ldots z, \Phi_{c_1^0}, [c_0^0, \Phi_{c_0^0}]\sigma\rangle$. If $v_1^0 = \varepsilon$ the automaton reaches this configuration right after reading c_0^0, skipping the previous one. Because $c_0^0 \doteq c_1^0$, a shift transition occurs, leading to configuration $\langle v_2^0 \ldots, \Phi_{v_2^0}, [c_1^0, \Phi_{c_0^0}]\sigma\rangle$. After reading the rest of u_0, the automaton comes to a configuration $\langle b_1 \ldots z, \Phi_{b_1}, [c_{m_0}^0, \Phi_{c_0^0}]\sigma\rangle$ (note that $m_0 = 0$ may hold as well). Since $c_{m_0}^0 \gtrdot b_1$, the topmost stack symbol is popped, leading to the configuration $\langle b_1 u_1 \ldots z, \Phi'_{b_1}, \sigma\rangle$. However, $\bigcirc^s_\chi \psi \in \Phi_{c_0^0}$, and due to rule 5, either $\bigcirc^s_\chi \psi \in \Phi'_{b_1}$ or $\bigcirc^{end}_\chi \psi \in \Phi'_{b_1}$. Here, the OPA nondeterministically guesses that this pop move is related to the end of a non-maximal chain, so $\bigcirc^s_\chi \psi \in \Phi'_{b_1}$ and $\bigcirc^{end}_\chi \psi \notin \Phi'_{b_1}$. The topmost stack symbol now contains the terminal symbol a, and $a \lessdot b_1$. So a push move brings the automaton to configuration $\langle u_1 \ldots z, \Phi_{u_1}, [b_1, \Phi'_{b_1}]\sigma\rangle$, with $\bigcirc^s_\chi \psi \in \Phi'_{b_1}$. Then, the automaton reads u_1 in a way similar to u_0. In particular, note that the stack symbol $[b_1, \Phi'_{b_1}]$ is not popped until the automaton reaches b_2, when a pop transition propagates $\bigcirc^s_\chi \psi$ from Φ'_{b_1} to the new state, due to another nondeterministic guess.

This process continues similarly for each b_k and u_k, until configuration $C_d = \langle dz, \Phi_d, [c, \Phi_{b_n}]\sigma\rangle$, where either $c = c^n_{m_n}$ or $c = b_n$, and $\bigcirc^s_\chi \psi \in \Phi_{b_n}$, is reached. Since in both cases $c \gtrdot d$, a pop transition brings the automaton to configuration $\langle dz, \Phi'_d, \sigma\rangle$, where $\bigcirc^s_\chi \psi \in \Phi_{b_n}$. This time, the OPA guesses that this is the end of a maximal chain, so we have $\bigcirc^{end}_\chi \psi \in \Phi'_d$

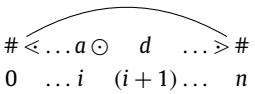
(by rule 5), and $\psi \in \Phi'_d$. Now, the topmost stack symbol is $[a, \Phi_a]$ or $[a, \Phi_f]$. Suppose $a \gtrdot d$: the next transition is a pop, which leads to $\langle dz, \Phi''_d, \gamma\rangle$ or $\langle dz, \Phi''_d, [f, \Phi_f]\gamma\rangle$, depending on the type of the move that previously read a. Due to rule 3, this transition can be performed iff ψ is also contained in Φ'_d. Note that subsequent pop transitions cannot drop ψ, and propagate it until the push or shift transition that reads d. They can, however, drop $\bigcirc^{end}_\chi \psi$, because it is in $\mathrm{Cl}_{aux}(\varphi)$, reaching a configuration in which neither $\bigcirc^{s}_\chi \psi$ nor $\bigcirc^{end}_\chi \psi$ is in the current state. In this case, either $\alpha' = \varepsilon$ or $\alpha' = [f, \Phi_f]$, respectively, so $|\alpha'| \leq 1$.

Conversely, if $a \doteq d$, then the next transition is a shift reading d, which leads to either $\langle z, \Phi''_d, [d, \Phi_f]\gamma\rangle$ or $\langle z, \Phi''_d, [d, \Phi_a][f, \Phi_f]\gamma\rangle$. This transition is defined, according to rule 3. In the former case, $\alpha' = [d, \Phi_f]$. For the latter note that $\bigcirc^{end}_\chi \psi$ can be nondeterministically dropped from Φ''_d during the shift. So, even in this case the automaton will eventually reach a configuration in which $[d, \Phi_a]$ is popped, and neither $\bigcirc^{end}_\chi \psi$ nor $\bigcirc^{s}_\chi \psi$ is present in the current state, fulfilling the thesis statement with $\alpha' = [f, \Phi_f]$. Note that this behavior agrees with Lemma 5.1.

Suppose the OPA takes the wrong guess that $\bigcirc_\chi \psi$ does not hold in i, and reads i from a configuration C'_0 in which $\bigcirc_\chi \psi \notin \Phi_a$. Then, $\bigcirc^{s}_\chi \psi$ is not inserted into the next state by rule 1 and propagated, so the OPA reaches C_d with $\bigcirc^{s}_\chi \psi \notin \Phi_{b_n}$. However, ψ is present in the next state and, by rule 3, also $\bigcirc^{end}_\chi \psi$. A subsequent pop transition would contradict rule 5, and the computation is blocked. This proves all accepting computations make the right guess, and pass through C_0.

[⇐] Suppose the OPA reads subword y with a computation as stated in the thesis. By contradiction, we also come to the conclusion that w is in the form of Fig. 11.

Indeed, if no chain started in i, then w would be of the form



where either $a \doteq d$ or $a \gtrdot d$. Suppose the OPA is in configuration $\langle adz, \Phi_a, \alpha\gamma\rangle$, with $a = \Phi_a \cap AP$, $\bigcirc_\chi \psi \in \Phi_a$, $\alpha = [f, \Phi_f]$, $f \in \mathscr{P}(AP)$, before reading position i. a is read by the automaton with either a push (if $f \lessdot a$) or a shift (if $f \doteq a$) move. Due to rules 1 and 2, it transitions to a configuration $\langle dz, \Phi_d, [a, \Phi_a][f, \Phi_f]\gamma\rangle$ or $\langle dz, \Phi_d, [a, \Phi_f]\gamma\rangle$, respectively, in both cases with $\bigcirc^{s}_\chi \psi \in \Phi_d$, and $\bigcirc^{end}_\chi \psi \notin \Phi_d$. If $a \doteq d$, then d is consumed by a shift transition. However, $\bigcirc^{s}_\chi \psi \in \Phi_d$, contradicting rule 4. So, the shift relation starting from state Φ_d is undefined. The same can be said if $a \gtrdot d$. Therefore, no computation of the automaton can read the subword $y = ad$ starting from a state containing $\bigcirc_\chi \psi$, which contradicts our hypothesis that such a computation exists. Thus, we can continue the proof with respect to the word structure of Fig. 11.

Let Φ be the state of the OPA before reading position i (as in the statement), and Θ the next one, pushed when reading the first input symbol of the body of the chain starting in i. For the automaton to reach a configuration in which the stack has the same length as before reading i plus at most 1, the stack symbol containing Θ must have been popped. Since, by hypothesis, $\bigcirc_\chi \psi \in \Phi$, by rule 1, $\bigcirc^{s}_\chi \psi$ must be in Θ. By rule 5, $\bigcirc^{s}_\chi \psi$ or $\bigcirc^{end}_\chi \psi$ must be present in the target state Ψ of that pop move. They can only be dropped by a subsequent shift or pop move, while a push would increase the stack size again. But, by rule 4, such a move can only occur if $\bigcirc^{s}_\chi \psi \notin \Psi$, so $\bigcirc^{end}_\chi \psi \in \Psi$. If a shift or a pop transition occurs, since the topmost stack symbol is the one containing the input symbol in i, it means i is in the $\doteq$ or $\gtrdot$ relation with the current position. So, the end of the maximal chain starting in i has been reached (cf. Lemma 5.1), and by rule 3 ψ holds in it, satisfying $\bigcirc_\chi \psi$ in i. □

5.3. Matching back ($\ominus_\chi$) operator

Here we report the additional rules needed to build an OPA accepting sentences satisfying $\ominus_\chi \psi$. Note that this construction is radically different from the one for the $\bigcirc_\chi$ operator, despite the two operators being the dual of each other. This is due to the asymmetric behavior of OPA, which read an input symbol with every push transition at the beginning of a chain, but read none during pop moves, at the end of chains.

If $\ominus_\chi \psi \in \mathrm{Cl}(\varphi)$, then $\psi \in \mathrm{Cl}(\varphi)$ and $\ominus^{s}_\chi \psi, \ominus^{end}_\chi \psi \in \mathrm{Cl}_{aux}(\varphi)$. For any $\Phi, \Theta, \Psi \in \mathrm{Atoms}(\varphi)$ and $a \in \mathscr{P}(AP)$, let $(\Phi, a, \Theta) \in \delta_{push} \cup \delta_{shift}$:

1. $\ominus_\chi \psi \in \Phi$ iff $\ominus^{s}_\chi \psi \in \Phi$;
2. $\ominus^{end}_\chi \psi \in \Theta$ iff $\psi \in \Phi$;
3. $\ominus^{s}_\chi \psi \notin \Theta$;

let $(\Phi, \Theta, \Psi) \in \delta_{pop}$:

4. $\ominus^{s}_\chi \psi \in \Psi$ iff $\ominus^{s}_\chi \psi \in \Theta$ or $\ominus^{end}_\chi \psi \in \Theta$;
5. $\ominus^{end}_\chi \psi \notin \Psi$.

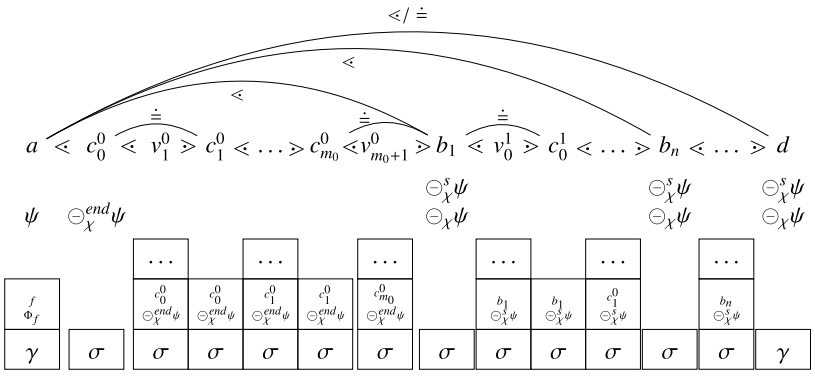


Fig. 13. Illustration of the execution of the automaton for $\ominus_\chi \psi$ on part of the OP word of Fig. 11, in the case $a \doteq d$ (top left of Fig. 11), or $a \lessdot d$ (not shown in Fig. 11). On the bottom, the stack contents before reading each terminal symbol are shown. Right above them, the relevant subformulas (such as $\ominus^s_\chi \psi$) are shown whenever they are contained in the state right before reading the terminal above.

Also, if $\Phi \in I$, then $\ominus_\chi \psi \notin \Phi$.

This construction exploits the fact that the last stack symbol to be popped before reading the right context of a backward-maximal chain is the one pushed when reading the first symbol of the body of the underlying simple chain (cf. Definition 2.5 and Lemma 5.1). Say $\ominus_\chi \psi$ holds in state $\Phi \in \text{Atoms}(\varphi)$, and $d \subseteq AP$ is the next symbol to be read, while $a \subseteq AP$ is the left context of the maximal chain whose right context is d: then ψ must hold in a. First, rule 1 enforces the auxiliary symbol $\ominus^s_\chi \psi$ in Φ, which is the target state of the last pop transition before d. By rule 4, $\ominus^s_\chi \psi$ or $\ominus^{end}_\chi \psi$ must be inserted into Θ, the stack symbol it pops. Θ is the state pushed when reading $b \subseteq AP$, the first symbol of the body of the simple chain underlying the maximal one. If $\ominus^{end}_\chi \psi \in \Theta$, b is the first symbol after a, and Θ is the target state of the push or shift transition reading a itself (in fact, by rule 5, Θ cannot be the target state of a pop). By rule 2, ψ must hold in the starting state of this transition, which is the one containing formulas that hold in a. If, instead, a and b are the contexts of an inner chain, we have $\ominus^s_\chi \psi \in \Theta$, and $\ominus^s_\chi \psi$ is propagated by rule 4 backwards, until it reaches the symbol right after a.

In practice, when ψ holds in a position, the OPA guesses whether that is the left context of a backward-maximal chain or not, by transitioning to a state containing $\ominus^{end}_\chi \psi$ or not. The guess is verified when reaching the right context of the chain, if any. The correctness of the constraints above is ensured by the following lemma.

Lemma 5.3. *Let AP be a finite set of atomic propositions, $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$ an OP alphabet, ψ an OPTL formula on it, and $\mathcal{A}_{\ominus_\chi \psi}$ an OPA satisfying rules 1-5 of Section 5.3. For any word $w = \#xyz\#$ on $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$, let position $j = |xy|$ in w: we have*

$$(w, j) \models \ominus_\chi \psi$$

if and only if $\mathcal{A}_{\ominus_\chi \psi}$ performs one computation that brings it from configuration $C_0 = \langle yz, \Phi, \alpha\gamma \rangle$ such that $\ominus^s_\chi \psi, \ominus^{end}_\chi \psi \notin \Phi$ to a configuration $\langle z, \Phi', \alpha'\gamma \rangle$ such that $|\alpha| = 1$, $|\alpha'| \leq 2$, and $\ominus_\chi \psi \in \Phi_j$, where Φ_j is the state of the automaton before reading j.

Moreover, if $(w, j) \models \ominus_\chi \psi$, then $\ominus_\chi \psi \in \Phi_j$ in all accepting computations of $\mathcal{A}_{\ominus_\chi \psi}$.

Proof. [$\Rightarrow$] If $\ominus_\chi \psi$ holds in j, then at least one chain must end there. Therefore, w must have the form of Figs. 11 and 15, except $a \lessdot d$ or $a \doteq d$ (but not $a \gtrdot d$). In the figures, i is the position such that $\overleftarrow{\chi}(i, j)$. The other chains possibly starting from i and ending in positions i_{b_k}, $1 \leq k \leq n$, such that $\overleftarrow{\chi}(i, i_{b_k})$, are also highlighted. Note that there might also be multiple chains ending in j, and in each one of the i_{b_k}s, forming the structure shown in Fig. 15. However, we do not need to analyze them in this proof, so the left AST of Fig. 11, where they are hidden in $v^n_{m_n+1}$, suffices.

Fig. 13 partially shows the behavior of the automaton we now describe. Suppose the automaton is in configuration $\langle a \dots z, \Phi_a, [f, \Phi_f]\gamma \rangle$, where $\alpha = [f, \Phi_f]$, and $\psi \in \Phi_a$. Depending on whether $f \doteq a$ or $f \lessdot a$, the next move is either a push or a shift. In both cases, due to rule 2, the next configuration is $\langle c^0_0 \dots z, \Phi_{c^0_0}, \sigma \rangle$ with $\ominus^{end}_\chi \psi \in \Phi_{c^0_0}$. Next, c^0_0 is read by a push, leading to $\langle v^0_1 \dots z, \Phi_{v^0_1}, [c^0_0, \Phi_{c^0_0}]\sigma \rangle$, with $\ominus^{end}_\chi \psi \in \Phi_{c^0_0}$. The state contained in the topmost stack symbol then remains untouched until configuration $\langle b_1 \dots z, \Phi_{b_1}, [c^0_{m_0}, \Phi_{c^0_0}]\sigma \rangle$ is reached (cf. proof of Lemma 5.2). Now, $c^0_{m_0} \gtrdot b_1$, and a pop move occurs: $\langle b_1 \dots z, \Phi'_{b_1}, \sigma \rangle$. Since $\ominus^{end}_\chi \psi \in \Phi_{c^0_0}$, all computations are such that $\bigcirc^s_\chi \psi \in \Phi'_{b_1}$, which satisfies rule 4. Then, b_1 is read by a push, leading to $\langle v^1_0 \dots z, \Phi_{v^1_0}, [b_1, \Phi'_{b_1}]\sigma \rangle$, and now $\ominus^s_\chi \psi$ is in the topmost stack symbol. By rule 1, this push may occur only if, by a nondeterministic guess, $\ominus_\chi \psi \in \Phi'_{b_1}$. This guess is correct, since b_1 also satisfies $\ominus_\chi \psi$.

The same reasoning can be extended to each b_k, $1 \leq k \leq n$, by replacing $\ominus^{end}_\chi \psi$ with $\ominus^s_\chi \psi$, proving that there exists a computation that leads the automaton to $\langle dz, \Phi_d, [c^n_{m_n}, \Phi_{b_n}]\sigma \rangle$, with $\ominus^s_\chi \psi \in \Phi_{b_n}$. A pop move occurs: $\langle dz, \Phi'_d, \sigma \rangle$. By rule 4, we have $\ominus^s_\chi \psi \in \Phi'_d$. Now, the symbol on top of the stack is a. Suppose $a \doteq d$: the next move is a shift leading to $\langle z, \Phi_z, [d, \Phi_{a|f}]\zeta \rangle$, with either $\Phi_{a|f} = \Phi_a$ and $\zeta = [f, \Phi_f]\gamma$, or $\Phi_{a|f} = \Phi_f$ and $\zeta = \gamma$, depending on the kind of move

reading a. In the former case, $\alpha' = [d, \Phi_a][f, \Phi_f]$, in the latter $\alpha' = [f, \Phi_f]$, which both satisfy the thesis statement. By rule 1, this shift may occur only if $\ominus_\chi \psi \in \Phi'_d$ due to a nondeterministic guess. If, instead, $a \lessdot d$, then the next move is a push. In this case, the proof is analogous, because push and shift transitions share the same rules. Note that it cannot be $a \gtrdot d$ (cf. Lemma 5.1), or the chain starting in a would not be the backward-maximal one ending in d.

By the above argument, it is also clear that any computation with $\psi \in \Phi_a$ must place $\ominus_\chi \psi$ in $\Phi_j = \Phi_d$, or it is blocked by rule 1, and cannot be accepting.

[$\Leftarrow$] If a computation as in the thesis statement exists, then position j must be the right context of at least one chain. By contradiction, suppose this is not true: w must be of the form shown below, with either $e \doteq d$ or $e \lessdot d$.

$$\begin{array}{ccccccccc} \# & \lessdot & \dots & e & \odot & d & \dots & \gtrdot & \# \\ 0 & & \dots (j-1) & & j & & \dots & & n \end{array}$$

The automaton reads e and d with two consecutive push or shift transitions. Suppose without loss of generality that they are both push moves (the rules for shift moves are the same). They bring the automaton from $\langle edz, \Phi_e, \sigma \rangle$ to $\langle dz, \Phi_d, [e, \Phi_e]\sigma \rangle$ and then to $\langle z, \Phi_z, [d, \Phi_d][e, \Phi_e]\sigma \rangle$. If $\ominus_\chi \psi \in \Phi_d$, then by rule 1 $\ominus^s_\chi \psi \in \Phi_d$. But Φ_d is the target state of the previous transition, which cannot be part of δ because it violates rule 3.

The fact that $\ominus_\chi \psi$ holds in j follows from the fact that the computation reads position j from a state containing $\ominus_\chi \psi$. Due to rule 1, every computation such that $\ominus_\chi \psi \in \Phi_j$ can go on reading j only if $\ominus^s_\chi \psi \in \Phi_j$. Also, we already proved that at least one chain ends in j. So the move reading j is preceded by a pop, which must remove from the stack a symbol whose state contains $\ominus^s_\chi \psi$ or $\ominus^{end}_\chi \psi$ (rule 4). Consider the move that pushed this symbol. If the symbol contains $\ominus^s_\chi \psi$, by rule 3 it must be preceded by a pop, again of a symbol containing $\ominus^s_\chi \psi$ or $\ominus^{end}_\chi \psi$. If the symbol contains $\ominus^{end}_\chi \psi$, by rule 5 the push must be preceded by another push or a shift move, so the beginning of the backward-maximal chain ending in j has been reached. Since this push/shift move brings the automaton to a state containing $\ominus^{end}_\chi \psi$, by rule 2 ψ holds, and so does $\ominus_\chi \psi$ in j. □

5.4. Summary until ($\mathcal{U}^\boxdot$) or since ($\mathcal{S}^\boxdot$) operator

For any $\boxdot \subseteq \{\lessdot, \doteq, \gtrdot\}$, if $\psi \, \mathcal{U}^\boxdot \, \theta \in \mathrm{Cl}(\varphi)$ then $\psi, \theta, \bigcirc_\chi(\psi \, \mathcal{U}^\boxdot \, \theta), \bigcirc(\psi \, \mathcal{U}^\boxdot \, \theta) \in \mathrm{Cl}(\varphi)$. Also, for any $\Phi \in \mathrm{Atoms}(\varphi)$, $\psi \, \mathcal{U}^\boxdot \, \theta \in \Phi$ iff either:

1. $\theta \in \Phi$; or
2. $\psi \in \Phi$ and $\bigcirc_\chi(\psi \, \mathcal{U}^\boxdot \, \theta) \in \Phi$; or
3. $\psi \in \Phi$ and $\bigcirc(\psi \, \mathcal{U}^\boxdot \, \theta) \in \Phi$.

If 1 holds, then $\psi \, \mathcal{U}^\boxdot \, \theta$ 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 $\boxdot$. This can be achieved by adding the following constraints: if only 3 holds in a state Φ and $\psi \, \mathcal{U}^\boxdot \, \theta \in \Phi$, 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 $\odot \in \boxdot$. 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 $\psi \, \mathcal{U}^\boxdot \, \theta \notin \Phi$, unless $\theta \in \Phi$.

Lemma 5.4. *Let AP be a finite set of atomic propositions, $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$ an OP alphabet, ψ and θ two OPTL formulae on it, $\boxdot \subseteq \{\lessdot, \doteq, \gtrdot\}$, and $\mathcal{A}_{\psi \mathcal{U}^\boxdot \theta}$ an OPA satisfying rules 1-5 of Section 5.2, and 1-3 of Section 5.4. For any word w on $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$, let position $i \leq |w|$ in w: we have*

$$(w, i) \models \psi \, \mathcal{U}^\boxdot \, \theta$$

if and only if $\mathcal{A}_{\psi \mathcal{U}^\boxdot \theta}$ performs one accepting computation such that $\psi \, \mathcal{U}^\boxdot \, \theta \in \Phi_i$, where Φ_i is the state of the automaton before reading position i.

Moreover, if $(w, i) \models \psi \, \mathcal{U}^\boxdot \, \theta$, then $\psi \, \mathcal{U}^\boxdot \, \theta \in \Phi_i$ in all accepting runs of $\mathcal{A}_{\psi \mathcal{U}^\boxdot \theta}$.

Proof. The validity of the expansion formulas used in rules 1 and 2 follows trivially from the definitions of OP summary path and until operator. As for rule 3, we need to prove that the additional constraint on push and shift transitions ensures the satisfaction of the requirements on precedence relations between consecutive positions.

If 1 or 2 hold in a position, then the truth of $\psi \, \mathcal{U}^\boxdot \, \theta$ is witnessed by at least one path not passing through the next position, and no additional constraint is necessary. Otherwise, if 3 holds in a state, the constraint on push and shift transitions blocks all computations where $\psi \, \mathcal{U}^\boxdot \, \theta$ would be witnessed by a path passing through consecutive positions in a precedence relation not in $\boxdot$.

Given the correctness of model checking for the $\bigcirc$ and $\bigcirc_\chi$ (Lemma 5.2) operators, it is possible to prove by induction on path length that all and only computations satisfying a weak version of $\psi\,\mathcal{U}^{\boxdot}\,\theta$ are not blocked before the end of the word. Moreover, the constraint on final states makes sure words where $\psi\,\mathcal{U}^{\boxdot}\,\theta$ is not eventually satisfied are not accepted. □

The construction for $\psi\,\mathcal{S}^{\boxdot}\,\theta$ is analogous: $\bigcirc_\chi$ and $\bigcirc$ are replaced with $\ominus_\chi$ and $\ominus$.

5.5. Yield-precedence hierarchical until ($\mathcal{U}^{\uparrow}$) operator

This construction exploits the automaton's stack to keep track of the requirements of the $\mathcal{U}^{\uparrow}$ operator in a way similar to $\bigcirc_\chi$. If $\psi\,\mathcal{U}^{\uparrow}\,\theta \in \mathrm{Cl}(\varphi)$ we introduce the auxiliary operators $\mathcal{U}_s^{\uparrow}$ and $\mathcal{U}_{end}^{\uparrow}$. We add $\psi, \theta \in \mathrm{Cl}(\varphi)$, and $\psi\,\mathcal{U}_s^{\uparrow}\,\theta, \psi\,\mathcal{U}_{end}^{\uparrow}\,\theta \in \mathrm{Cl}_{aux}(\varphi)$.
Given any $\Phi, \Theta, \Psi \in \mathrm{Atoms}(\varphi)$ and $a \in \mathscr{P}(AP)$, let $(\Phi, a, \Theta) \in \delta_{push} \cup \delta_{shift}$, then

1. $\psi\,\mathcal{U}^{\uparrow}\,\theta \in \Phi$ iff $\psi\,\mathcal{U}_s^{\uparrow}\,\theta \in \Theta$;

let $(\Phi, \Theta, \Psi) \in \delta_{pop}$ or $(\Phi, a, \Theta) \in \delta_{shift}$: then

2. $\psi\,\mathcal{U}_s^{\uparrow}\,\theta \notin \Phi$ and $\psi\,\mathcal{U}_{end}^{\uparrow}\,\theta \notin \Phi$;

let $(\Phi, \Theta, \Psi) \in \delta_{pop}$: then

3. $\psi\,\mathcal{U}_s^{\uparrow}\,\theta \in \Theta$ iff, either, $\theta \in \Psi$ or ($\psi \in \Psi$, $\theta \notin \Psi$ and $\psi\,\mathcal{U}_s^{\uparrow}\,\theta \in \Psi$);
4. if $\theta \in \Psi$ and $\psi\,\mathcal{U}_s^{\uparrow}\,\theta \in \Theta$, then $\psi\,\mathcal{U}_{end}^{\uparrow}\,\theta \in \Psi$.

States containing $\psi\,\mathcal{U}_s^{\uparrow}\,\theta$ or $\psi\,\mathcal{U}_{end}^{\uparrow}\,\theta$ are excluded from the final set F.

Similarly to what happens for $\bigcirc_\chi$, the auxiliary operator $\psi\,\mathcal{U}_s^{\uparrow}\,\theta$ is put into the state after the one in which $\psi\,\mathcal{U}^{\uparrow}\,\theta$ holds by rule 1. Then, the state containing $\psi\,\mathcal{U}_s^{\uparrow}\,\theta$ is pushed on stack. Each time it is popped, we check if either θ holds in the next state, or if ψ holds and the path continues, so $\psi\,\mathcal{U}_s^{\uparrow}\,\theta$ must continue being propagated (rule 3). In the former case, $\psi\,\mathcal{U}_{end}^{\uparrow}\,\theta$ is put into the next state to mark the end of the hierarchical path (rule 4). Let j be the right context of the maximal chain starting where $\psi\,\mathcal{U}^{\uparrow}\,\theta$ is supposed to hold. The purpose of rule 2 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 right contexts of non-maximal chains are).

The correctness of this construction is shown in the following lemma.

Lemma 5.5. *Let AP be a finite set of atomic propositions, $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$ an OP alphabet, ψ and θ two OPTL formulae on it, and $\mathcal{A}_{\psi\mathcal{U}^{\uparrow}\theta}$ an OPA satisfying rules 1-4 of Section 5.5. For any word $w = \#xyz\#$ on $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$, let position $i = |x| + 1$ in w. We have*

$$(w, i) \models \psi\,\mathcal{U}^{\uparrow}\,\theta$$

if and only if $\mathcal{A}_{\psi\mathcal{U}^{\uparrow}\theta}$ performs one computation that brings it from configuration $C_0 = \langle yz, \Phi, \alpha\gamma\rangle$ with $\psi\,\mathcal{U}^{\uparrow}\,\theta \in \Phi$ to a configuration $\langle z, \Phi', \alpha'\gamma\rangle$ such that $|\alpha| = 1$, $|\alpha'| \leq 1$, and $\psi\,\mathcal{U}_s^{\uparrow}\,\theta, \psi\,\mathcal{U}_{end}^{\uparrow}\,\theta \notin \Phi'$.

Moreover, if $(w, i) \models \psi\,\mathcal{U}^{\uparrow}\,\theta$, all accepting computations of $\mathcal{A}_{\psi\mathcal{U}^{\uparrow}\theta}$ pass through C_0.

Proof. This correctness argument is largely similar to the one of the proof of Lemma 5.2. For $\psi\,\mathcal{U}^{\uparrow}\,\theta$ to hold in i, w must have the structure shown in Fig. 11. Indeed, if $(w, i) \models \psi\,\mathcal{U}^{\uparrow}\,\theta$, then at least two chains must start in i.

If this is not the case, the word is rejected by the automaton. Suppose, by contradiction, $w = \# \lessdot x \odot_x a \odot_d d \odot_z z \gtrdot \#$, with no chain starting in a, and either $a \doteq d$ or $a \gtrdot d$. Let Φ_a be the state of the automaton before reading a, with $\psi\,\mathcal{U}^{\uparrow}\,\theta \in \Phi_a$. By rule 1, after reading a the automaton is in state Φ_d, with $\psi\,\mathcal{U}_s^{\uparrow}\,\theta \in \Phi_d$. If $a \doteq d$ (resp. $a \gtrdot d$), then the next transition is a shift (resp. pop), but due to rule 2 it cannot occur.

We must now rule out the possibility that only one single chain starts in i, and we do so by contradiction. The behavior of the automaton in this case is the same it would be while reading the first part of a generic structure such as the one in Fig. 11. Suppose the automaton is in configuration $\langle a \ldots z, \Phi_a, [f, \Phi_f]\gamma\rangle$ with $\alpha = [f, \Phi_f]$, and it guesses that $\psi\,\mathcal{U}^{\uparrow}\,\theta$ holds in i, so $\psi\,\mathcal{U}^{\uparrow}\,\theta \in \Phi_a$. If $f \doteq a$, then the next configuration is $\langle c_0^0 \ldots z, \Phi_{c_0^0}, [a, \Phi_f]\gamma\rangle$, and $\langle c_0^0 \ldots z, \Phi_{c_0^0}, [a, \Phi_a][f, \Phi_f]\gamma\rangle$ if $f \lessdot a$. In both cases, $\psi\,\mathcal{U}_s^{\uparrow}\,\theta \in \Phi_{c_0^0}$ because of rule 1. From now on, we will consider either $\sigma = [a, \Phi_f]\gamma$ or $\sigma = [a, \Phi_a][f, \Phi_f]\gamma$, depending on the precedence relation between f and a. Afterwards, the automaton reads c_0^0 and transitions to $\langle v_1^0 \ldots z, \Phi_{v_1^0}, [c_0^0, \Phi_{c_0^0}]\sigma\rangle$, with $\psi\,\mathcal{U}_s^{\uparrow}\,\theta \in \Phi_{c_0^0}$. The automaton then goes on reading the rest of u_0, leaving $\Phi_{c_0^0}$ on the stack.

Now, suppose $n = 0$, i.e. the only chain starting in i is the one ending with d (and there are no b_ks). After reading $v^0_{m_0+1}$, the automaton is in configuration $\langle dz, \Phi_d, [c^0_{m_0}, \Phi_{c^0_0}]\sigma\rangle$. Since $c^0_{m_0} \gtrdot d$, the topmost symbol is popped, leading to $\langle dz, \Phi'_d, \sigma\rangle$. Due to rule 3, this transition can only occur if $\psi \in \Phi_d$ or $\theta \in \Phi_d$. If this is the case, then either $\psi\, \mathcal{U}^{\uparrow}_s\, \theta \in \Phi'_d$ (rule 3) or $\psi\, \mathcal{U}^{\uparrow}_{end}\, \theta \in \Phi'_d$ (rule 4). The next transition is a pop if $a \gtrdot d$, or a shift if $a \doteq d$. In both cases, the computation stops due to rule 2. This is consistent with the fact that if there are no non-maximal chains starting in a, then $\psi\, \mathcal{U}^{\uparrow}\, \theta$ must be false. So w has the structure of Fig. 11, with $n > 0$.

[$\Rightarrow$] Let us go back to the configuration of the automaton after reading u_0, i.e. $\langle b_1 \dots z, \Phi_{b_1}, [c^0_{m_0}, \Phi_{c^0_0}]\sigma\rangle$. Since $c^0_{m_0} \gtrdot b_1$, a pop transition occurs, leading to $\langle b_1 \dots z, \Phi'_{b_1}, \sigma\rangle$. Because $\psi\, \mathcal{U}^{\uparrow}_s\, \theta \in \Phi_{c^0_0}$, rule 3 applies. If $\theta \in \Phi_{b_1}$, i.e. θ holds in i_{b_1}, then $\psi\, \mathcal{U}^{\uparrow}\, \theta$ is satisfied in i by the hierarchical path made of position i_{b_1} only, and $\psi\, \mathcal{U}^{\uparrow}_{end}\, \theta$ is placed in Φ'_{b_1}. Instead, if $\psi \in \Phi_{b_1}$, then the hierarchical path must continue with other b_p positions, $1 < p \leq n$. So, $\psi\, \mathcal{U}^{\uparrow}_s\, \theta \in \Phi'_{b_1}$. The transition reading b_1 is a push: $\langle u_1 \dots z, \Phi_{u_1}, [b_1, \Phi'_{b_1}]\sigma\rangle$. Then, while the automaton reads u_1, it never pops the stack symbol containing Φ'_{b_1}, until b_2 is reached. This process goes on until d is reached, in the same way for each b_p.

If $\psi\, \mathcal{U}^{\uparrow}\, \theta$ holds in i, then by its semantics there exists a value k, $1 \leq k \leq n$, such that θ holds in i_{b_k} and ψ holds in i_{b_h} for each $1 \leq h < k$. Before reading i_{b_k}, the automaton is in configuration $C_{b_k} = \langle b_k \dots z, \Phi_{b_k}, [c^{k-1}_{m_{k-1}}, \Phi_{b_{k-1}}]\sigma\rangle$, with $\psi\, \mathcal{U}^{\uparrow}_s\, \theta \in \Phi_{b_{k-1}}$. Since $\theta \in \Phi_{b_k}$, due to rules 3 and 4 it reaches $\langle b_k \dots z, \Phi'_{b_k}, \sigma\rangle$, with $\psi\, \mathcal{U}^{\uparrow}_{end}\, \theta \in \Phi'_{b_k}$, and $\psi\, \mathcal{U}^{\uparrow}_s\, \theta \notin \Phi'_{b_k}$. The computation goes on until $\langle dz, \Phi_d, [c^n_{m_n}, \Phi_{c^n_0}]\sigma\rangle$ is reached, no matter whether ψ or θ hold in i_h, with $h > k$. Since $\psi\, \mathcal{U}^{\uparrow}_s\, \theta, \psi\, \mathcal{U}^{\uparrow}_{end}\, \theta \notin \Phi_{c^n_0}$, the automaton can read d normally, transitioning to $\langle z, \Phi_z, \beta\gamma\rangle$, where $\psi\, \mathcal{U}^{\uparrow}_s\, \theta, \psi\, \mathcal{U}^{\uparrow}_{end}\, \theta \notin \Phi_z$. It is easy to show that either $\beta = \varepsilon$, $\beta = [d, \Phi_f]$, $\beta = [d, \Phi_a]$, or $\beta = [d, \Phi_a][f, \Phi_f]$. In the first three cases, $\alpha' = \beta$. In the last one, $[d, \Phi_a]$ will eventually be popped, fulfilling the thesis statement with $\alpha' = [f, \Phi_f]$.

Suppose the OPA takes the wrong guess that $\psi\, \mathcal{U}^{\uparrow}\, \theta$ does not hold in i, so $\psi\, \mathcal{U}^{\uparrow}\, \theta \notin \Phi_a$. Then, it arrives in configuration C_{b_k} with $\psi\, \mathcal{U}^{\uparrow}_s\, \theta \notin \Phi_{c^k_0}$. Since, however, θ holds in b_k, $\theta \in \Phi_{b_k}$, and by rule 3 the next pop move cannot occur, ending the computation. Thus, for a run to be accepting, the right guess must be made when reaching C_0.

[$\Leftarrow$] We already showed the word must have the form of Fig. 11. Note that, for the computation to continue after reading d, there must exist a value k, $1 \leq k \leq n$, such that $\theta \in \Phi_{b_k}$ and $\psi \in \Phi_{b_h}$ for each $1 \leq h < k$, where Φ_{b_p} is the state of the automaton before reading b_p. In fact, suppose there exists no such k. $\psi\, \mathcal{U}^{\uparrow}_s\, \theta$ is introduced in the state contained in the first stack symbol pushed after reading i by rule 1. We already saw that, due to rule 3, the computation stops if a state containing $\psi\, \mathcal{U}^{\uparrow}_s\, \theta$ is popped from stack and neither ψ nor θ hold in the previous state. Moreover, if ψ is present in one of such states, $\psi\, \mathcal{U}^{\uparrow}_s\, \theta$ is propagated in the next stack symbol, and so are the requirements of rule 3. This can only stop if θ is found in one of these positions. So, in this case $\psi \in \Phi_{b_h}$ for all $1 \leq h \leq n$. By correctness of model checking for ψ and θ, this implies θ holds in i_{b_k} and ψ holds in all i_{b_h} for $1 \leq h \leq n$. So, $\psi\, \mathcal{U}^{\uparrow}\, \theta$ holds in i. Otherwise, the automaton eventually reaches configuration $\langle dz, \Phi_d, [c^n_{m_n}, \Phi_{c^n_0}]\sigma\rangle$, with $\psi\, \mathcal{U}^{\uparrow}_s\, \theta \in \Phi_{c^n_0}$, and the computation is blocked by rule 2. □

5.6. Yield-precedence hierarchical since ($\mathcal{S}^{\downarrow}$) operator

If $\psi\, \mathcal{S}^{\downarrow}\, \theta \in \Phi$, with $\Phi \in \mathrm{Atoms}(\varphi)$, then $\psi, \theta \in \mathrm{Cl}(\varphi)$ and $\psi\, \mathcal{S}^{\downarrow}_s\, \theta, \psi\, \mathcal{S}^{\downarrow}_{end}\, \theta, \bigcirc_\chi(\psi\, \mathcal{S}^{\downarrow}_s\, \theta) \in \mathrm{Cl}_{aux}(\varphi)$. Also, $\psi\, \mathcal{S}^{\downarrow}\, \theta \in \Phi$ iff $\bigcirc_\chi(\psi\, \mathcal{S}^{\downarrow}_s\, \theta) \in \Phi$, introducing the auxiliary symbol $\psi\, \mathcal{S}^{\downarrow}_s\, \theta$. For any $\Phi, \Theta, \Psi \in \mathrm{Atoms}(\varphi)$, let $(\Phi, \Theta, \Psi) \in \delta_{pop}$:

1. $\psi\, \mathcal{S}^{\downarrow}_s\, \theta \in \Psi$ iff, either, ($\theta \in \Theta$ and $\psi\, \mathcal{S}^{\downarrow}_{end}\, \theta \in \Theta$), or ($\psi \in \Theta$, $\theta \notin \Theta$ and $\psi\, \mathcal{S}^{\downarrow}_s\, \theta \in \Theta$);
2. if $\theta \in \Psi$, then $\psi\, \mathcal{S}^{\downarrow}_{end}\, \theta \in \Psi$;

for any $a \in \mathscr{P}(AP)$, let $(\Phi, a, \Theta) \in \delta_{push} \cup \delta_{shift}$:

3. then $\psi\, \mathcal{S}^{\downarrow}_s\, \theta \notin \Theta$ and $\psi\, \mathcal{S}^{\downarrow}_{end}\, \theta \notin \Theta$.

The idea behind this definition is the following. Suppose $\psi\, \mathcal{S}^{\downarrow}\, \theta$ holds in a state $\Phi \in \mathrm{Atoms}(\varphi)$, associated to the terminal symbol $a \in \mathscr{P}(\varphi)$. Then the auxiliary operator $\psi\, \mathcal{S}^{\downarrow}_s\, \theta$ is enforced through a $\bigcirc_\chi$ operator into the state of the OPA before it reads the right context of the forward-maximal chain starting in a. This state is the result of a pop move that pops the stack symbol pushed when reading the right context of a non-maximal chain starting in a, i.e. a position that may be part of a yield-precedence hierarchical path. Every time the stack symbol related to such a position is popped from stack, rule 1 ensures that either θ holds in the next position in the path, which ends there with $\psi\, \mathcal{S}^{\downarrow}_{end}\, \theta$, or ψ holds in there, along with $\psi\, \mathcal{S}^{\downarrow}_s\, \theta$, which ensures the prosecution of the path. Rule 3 prevents the first position right after a from being considered part of the path, preventing its prosecution with $\psi\, \mathcal{S}^{\downarrow}_s\, \theta$ and $\psi\, \mathcal{S}^{\downarrow}_{end}\, \theta$. Note that the acceptance conditions of $\bigcirc_\chi(\psi\, \mathcal{S}^{\downarrow}_s\, \theta)$ already ensure the satisfaction of formula $\psi\, \mathcal{S}^{\downarrow}\, \theta$ before the end of the string.

Lemma 5.6. *Let AP be a finite set of atomic propositions, $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$ an OP alphabet, ψ and θ two OPTL formulae on it, and $\mathcal{A}_{\psi\,\mathcal{S}^{\downarrow}\theta}$ an OPA satisfying rules 1-3 of Section 5.6, and 1-5 of Section 5.2. For any word $w = \#xyz\#$ on $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$, let position $i = |x| + 1$ in w: we have*

$$(w, i) \models \psi\ \mathcal{S}^{\downarrow}\ \theta$$

if and only if $\mathcal{A}_{\psi\,\mathcal{S}^{\downarrow}\theta}$ performs one computation that brings it from configuration $C_0 = \langle yz, \Phi, \alpha\gamma\rangle$ with $\psi\,\mathcal{S}^{\downarrow}\,\theta \in \Phi$ to a configuration $\langle z, \Phi', \alpha'\gamma\rangle$ such that $\psi\ \mathcal{S}_s^{\downarrow}\ \theta, \psi\ \mathcal{S}_{end}^{\downarrow}\ \theta \notin \Phi'$, $|\alpha| = 1$, and $|\alpha'| \leq 1$.

Moreover, if $(w, i) \models \psi\ \mathcal{S}^{\downarrow}\ \theta$, all accepting computations of $\mathcal{A}_{\psi\,\mathcal{S}^{\downarrow}\theta}$ go through C_0.

Proof. [$\Rightarrow$] Suppose $(w, i) \models \psi\,\mathcal{S}^{\downarrow}\,\theta$. Then, i is the left context of at least two nested chains. Also, there exists $1 \leq k \leq n$ such that $(w, i_{b_k}) \models \theta$ and for all $k < p \leq n$, $(w, i_{b_p}) \models \psi \wedge \neg\theta$. Before reading a, the OPA is in configuration $C_0 = \langle a \ldots z, \Phi_a, \gamma\rangle$, and it guesses that $\psi\ \mathcal{S}^{\downarrow}\ \theta$ holds in i, so $\psi\ \mathcal{S}^{\downarrow}\ \theta \in \Phi_a$ and $\bigcirc_\chi(\psi\ \mathcal{S}_s^{\downarrow}\ \theta) \in \Phi_a$. Then, it reads the string normally until it is in configuration $\langle b_k \ldots z, \Phi_{b_k}, \sigma\rangle$, before reading b_k with a push move. By hypothesis, we have $\theta \in \Phi_{b_k}$. Φ_{b_k} is the state resulting from a pop move, so by rule 2 we have $\psi\ \mathcal{S}_{end}^{\downarrow}\ \theta \in \Phi_{b_k}$. Then, the push transition brings the automaton to $\langle v_0^k \ldots z, \Phi_{v_0^k}, [b_k, \Phi_{b_k}]\sigma\rangle$.

Let $\langle b_p \ldots z, \Phi_{b_p}, [c_{m_{p-1}}^{p-1}, \Phi_{b_{p-1}}]\sigma\rangle$ be the configuration of the OPA when reaching b_p, for any $k < p \leq n$. If $p = k+1$, then $\theta \in \Phi_{b_{p-1}}$ and $\psi\ \mathcal{S}_{end}^{\downarrow}\ \theta \in \Phi_{b_{p-1}}$. Otherwise, it is possible to inductively show that $\psi \in \Phi_{b_{p-1}}$, $\theta \notin \Phi_{b_{p-1}}$ and $\psi\ \mathcal{S}_s^{\downarrow}\ \theta \in \Phi_{b_{p-1}}$ By rule 1, in both cases a pop move leads to $\langle b_p \ldots z, \Phi'_{b_p}, \sigma\rangle$ with $\psi\ \mathcal{S}_s^{\downarrow}\ \theta \in \Phi'_{b_p}$. Note that, by hypothesis, $\psi \in \Phi'_{b_p}$. Again, Φ'_{b_p} is pushed, and the run proceeds until d is reached.

The configuration before reading d is $\langle dz, \Phi_d, [c, \Phi_{b_n}]\sigma\rangle$, with either $c = c_{m_n}^n$ or $c = b_n$, $\psi \in \Phi_{b_n}$, $\theta \notin \Phi_{b_n}$, and $\psi\ \mathcal{S}_s^{\downarrow}\ \theta \in \Phi_{b_n}$. By rule 1, a pop move leads to $\langle dz, \Phi'_d, \sigma\rangle$, with $\psi\ \mathcal{S}_s^{\downarrow}\ \theta \in \Phi_d$. This is consistent with $\bigcirc_\chi(\psi\ \mathcal{S}_s^{\downarrow}\ \theta) \in \Phi_a$, which verifies the initial guess. By Lemma 5.2, if $\psi\ \mathcal{S}_s^{\downarrow}\ \theta \in \Phi_d$ all accepting computations have $\bigcirc_\chi(\psi\ \mathcal{S}_s^{\downarrow}\ \theta) \in \Phi_a$, so they all make the right guess, and pass through C_0. Moreover, there exists a computation that after reading d reaches a state that contains no auxiliary symbols related to $\psi\,\mathcal{S}^{\downarrow}\,\theta$, and only the topmost stack symbol has changed w.r.t. before i was read.

[$\Leftarrow$] For the other side of the implication, the correctness of $\bigcirc_\chi(\psi\ \mathcal{S}_s^{\downarrow}\ \theta)$, which must hold in i, implies at least one chain starts in i. For the second chain, notice that in a word such as $\# \lessdot x \odot_x a \lessdot c_0^0 v_1^0 \ldots c_{m_0}^0 v_{m_0+1}^0 \gtrdot d \odot_z z \gtrdot \#$, c_0^0 is read by a push transition that stores a state $\Phi_{c_0^0}$ on stack. Due to rule 3, $\psi\ \mathcal{S}_s^{\downarrow}\ \theta, \psi\ \mathcal{S}_{end}^{\downarrow}\ \theta \notin \Phi_{c_0^0}$. Then, $\Phi_{c_0^0}$ is popped when the automaton reaches d. However, since $\bigcirc_\chi(\psi\ \mathcal{S}_s^{\downarrow}\ \theta)$ holds in a, by Lemma 5.2 we have $\psi\ \mathcal{S}_s^{\downarrow}\ \theta \in \Phi'_d$ (i.e. the state of the OPA before reading d). So, the last pop move is of the form $(\Phi_d, \Phi_{c_0^0}, \Phi'_d)$ with $\psi\ \mathcal{S}_s^{\downarrow}\ \theta, \psi\ \mathcal{S}_{end}^{\downarrow}\ \theta \notin \Phi_{c_0^0}$ and $\psi\ \mathcal{S}_s^{\downarrow}\ \theta \in \Phi'_d$, which contradicts rule 1. Thus, by contradiction, the structure of w is the one of Fig. 11.

In this case, too, there exists a value $1 \leq k \leq n$ such that $\theta \in \Phi_{b_k}$ and for all $k < p \leq n$, $\psi \in \Phi_{b_p}$. This is justified by the fact that $\bigcirc_\chi(\psi\ \mathcal{S}_s^{\downarrow}\ \theta) \in \Phi_a$, and by correctness of the $\bigcirc_\chi$ operator we have $\psi\ \mathcal{U}_s^{\uparrow}\ \theta \in \Phi_d$. Since d is the right context of a chain, the transition leading to Φ_d is a pop. By rule 1, the state Φ_{b_n} popped from stack either contains θ and $\psi\ \mathcal{S}_{end}^{\downarrow}\ \theta$, and $k = n$, or ψ and $\psi\ \mathcal{S}_s^{\downarrow}\ \theta$, so $k < n$. In the latter case, since the push move reading each b_p, $k < p \leq n$, is preceded by a pop, rule 1 applies. So $\psi\ \mathcal{S}_s^{\downarrow}\ \theta$ is propagated backwards, with ψ holding in each b_p, until b_k is reached, and the first case of rule 1 applies, with θ holding in b_k. This must happen, or $\psi\ \mathcal{S}_s^{\downarrow}\ \theta$ appears in $\Phi_{c_0^0}$, the state of the OPA after reading position i, which contradicts rule 3. The existence of k proves that $\psi\,\mathcal{U}^{\uparrow}\,\theta$ holds in i. □

5.7. Take-precedence hierarchical until ($\mathcal{U}^{\downarrow}$) operator

Let $\Phi \in \text{Atoms}(\varphi)$, with $\psi\,\mathcal{U}^{\downarrow}\,\theta \in \Phi$: then $\psi, \theta, \ominus\psi, \ominus\theta, \ominus_\chi\psi, \ominus_\chi\theta, \ominus_\chi\top \in \text{Cl}(\varphi)$ and $\psi\,\mathcal{U}_s^{\downarrow}\,\theta, \psi\,\mathcal{U}_{end}^{\downarrow}\,\theta \in \text{Cl}_{aux}(\varphi)$. For any $\Phi, \Theta \in \text{Atoms}(\varphi)$ and $a \in \mathscr{P}(AP)$, let $(\Phi, a, \Theta) \in \delta_{push} \cup \delta_{shift}$: then

1. $\psi\ \mathcal{U}_s^{\downarrow}\ \theta \notin \Theta$ and $\psi\ \mathcal{U}_{end}^{\downarrow}\ \theta \notin \Theta$;
2. $\psi\,\mathcal{U}^{\downarrow}\,\theta \in \Phi$ iff $\psi\ \mathcal{U}_{end}^{\downarrow}\ \theta \in \Phi$.

Moreover, for any $\Phi, \Theta, \Psi \in \text{Atoms}(\varphi)$, let $(\Phi, \Theta, \Psi) \in \delta_{pop}$:

3. $\psi\ \mathcal{U}_s^{\downarrow}\ \theta \in \Phi$ iff $\psi\ \mathcal{U}_{end}^{\downarrow}\ \theta \in \Psi$;
4. $\psi\ \mathcal{U}_s^{\downarrow}\ \theta \in \Psi$ iff, either, $\ominus_\chi\theta \vee (\neg\ominus_\chi\top \wedge \ominus\theta) \in \Theta$, or $(\ominus_\chi\psi \vee (\neg\ominus_\chi\top \wedge \ominus\psi) \in \Theta$ and $\psi\ \mathcal{U}_s^{\downarrow}\ \theta \in \Phi)$.

Of course, if $\psi\,\mathcal{U}^{\downarrow}\,\theta \in \Phi$ then $\Phi \notin I$.

	input	state	stack	rel.	move	rule
1	$\{\mathbf{b}, \mathrm{p}_1\} \lessdot \{\mathbf{b}, \mathrm{p}_2\} \lessdot \mathbf{b} \gtrdot \mathbf{d} \gtrdot \#$	$q_0 = \{\mathbf{b}, \mathrm{p}_1, \bigcirc_\chi(\mathrm{p}_1\, \mathcal{U}^{\downarrow}\, \mathrm{p}_2)\}$	$\bot$	$\# \lessdot \mathbf{b}$	push	–
2	$\{\mathbf{b}, \mathrm{p}_2\} \lessdot \mathbf{b} \gtrdot \mathbf{d} \gtrdot \#$	$q_1 = \begin{Bmatrix} \mathbf{b}, \mathrm{p}_2, \ominus \mathrm{p}_1, \\ \bigcirc^s_\chi(\mathrm{p}_1\, \mathcal{U}^{\downarrow}\, \mathrm{p}_2) \end{Bmatrix}$	$[\{\mathbf{b}, \mathrm{p}_1\}, q_0]\bot$	$\mathbf{b} \lessdot \mathbf{b}$	push	–
3	$\mathbf{b} \gtrdot \mathbf{d} \gtrdot \#$	$q_2 = \{\mathbf{b}, \ominus \mathrm{p}_2\}$	$[\{\mathbf{b}, \mathrm{p}_2\}, q_1][\{\mathbf{b}, \mathrm{p}_1\}, q_0]\bot$	$\mathbf{b} \lessdot \mathbf{b}$	push	–
4	$\mathbf{d} \gtrdot \#$	$\{\mathbf{d}, \mathrm{p}_1\, \mathcal{U}^{\downarrow}\, \mathrm{p}_2\}$	$[\mathbf{b}, q_2][\{\mathbf{b}, \mathrm{p}_2\}, q_1][\{\mathbf{b}, \mathrm{p}_1\}, q_0]\bot$	$\mathbf{b} \gtrdot \mathbf{d}$	pop	4
5	$\mathbf{d} \gtrdot \#$	$\{\mathbf{d}, \mathrm{p}_1\, \mathcal{U}^{\downarrow}\, \mathrm{p}_2, \mathrm{p}_1\, \mathcal{U}^{\downarrow}_s\, \mathrm{p}_2\}$	$[\{\mathbf{b}, \mathrm{p}_2\}, q_1][\{\mathbf{b}, \mathrm{p}_1\}, q_0]\bot$	$\mathbf{b} \gtrdot \mathbf{d}$	pop	4
6	$\mathbf{d} \gtrdot \#$	$\begin{Bmatrix} \mathbf{d}, \mathrm{p}_1\, \mathcal{U}^{\downarrow}\, \mathrm{p}_2, \mathrm{p}_1\, \mathcal{U}^{\downarrow}_s\, \mathrm{p}_2, \\ \mathrm{p}_1\, \mathcal{U}^{\downarrow}_{end}\, \mathrm{p}_2, \bigcirc^{end}_\chi(\mathrm{p}_1\, \mathcal{U}^{\downarrow}\, \mathrm{p}_2) \end{Bmatrix}$	$[\{\mathbf{b}, \mathrm{p}_1\}, q_0]\bot$	$\mathbf{b} \gtrdot \mathbf{d}$	pop	3
7	$\mathbf{d} \gtrdot \#$	$q_3 = \begin{Bmatrix} \mathbf{d}, \mathrm{p}_1\, \mathcal{U}^{\downarrow}\, \mathrm{p}_2, \\ \mathrm{p}_1\, \mathcal{U}^{\downarrow}_{end}\, \mathrm{p}_2 \end{Bmatrix}$	$\bot$	$\# \lessdot \mathbf{d}$	push	2
8	#	{#}	$[\mathbf{d}, q_3]\bot$	$\mathbf{d} \gtrdot \#$	pop	–
9	#	{#}	$\bot$	$\# \doteq \#$	–	–

Fig. 14. A run of the automaton for formula $\bigcirc_\chi(\mathrm{p}_1\, \mathcal{U}^{\downarrow}\, \mathrm{p}_2)$ accepting string "$\{\mathbf{b}, \mathrm{p}_1\}\{\mathbf{b}, \mathrm{p}_2\}\mathbf{bd}$" w.r.t. an OPM in which $\mathbf{b} \lessdot \mathbf{b}$ and $\mathbf{b} \gtrdot \mathbf{d}$. (Braces are omitted in singletons.)

Fig. 14 shows an example accepting run for a formula containing this operator. The structure of the word read by the OPA is an instance of that of Fig. 15, with multiple chains (starting in **b**-positions in the example) ending in the same position j (**d** in the example). Let $q_3 \in \mathrm{Atoms}(\varphi)$ be the state of the automaton just before reading **d**. The consistency of $\mathrm{p}_1\, \mathcal{U}^{\downarrow}\, \mathrm{p}_2 \in q_3$ is checked during the *pop* transitions that remove from the automaton's stack the symbols related to all chains of which **d** is the right context. The auxiliary operator $\mathcal{U}^{\downarrow}_s$ is kept in the current state until the position in which $\mathrm{p}_1\, \mathcal{U}^{\downarrow}\, \mathrm{p}_2$ holds, when its requirements are finally checked. First, the OPA guesses that $\mathrm{p}_1\, \mathcal{U}^{\downarrow}\, \mathrm{p}_2$ holds in **d**, introducing it in step 4. This guess is verified by rule 2 in step 7, with the auxiliary operator $\mathrm{p}_1\, \mathcal{U}^{\downarrow}_{end}\, \mathrm{p}_2$. By rule 3 the pop transition between steps 6-7 requires $\mathrm{p}_1\, \mathcal{U}^{\downarrow}_s\, \mathrm{p}_2$ in the previous state. Constraint 4 checks that a path satisfying $\mathrm{p}_1\, \mathcal{U}^{\downarrow}\, \mathrm{p}_2$ actually exists. All previous consecutive *pop* transitions must either enforce p_2 in the left context of the chain, i_{b_k} (with label **b**), by having $\ominus \mathrm{p}_2$ in the popped stack symbol, and letting the path end (cf. step 4-5), or they enforce p_1 in i_{b_k}, and let the path continue by leaving $\mathrm{p}_1\, \mathcal{U}^{\downarrow}_s\, \mathrm{p}_2$ in the previous state (cf. step 5-6). In order to disallow paths that do not end with a position in which p_2 holds, by rule 1, $\mathrm{p}_1\, \mathcal{U}^{\downarrow}_s\, \mathrm{p}_2$ cannot be the target state of any *push* or *shift* transition, and in particular of the one reading the terminal symbol before **d**, which cannot be part of the path.

The correctness of these constraints is assessed by the following lemma.

Lemma 5.7. *Let AP be a finite set of atomic propositions, $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$ an OP alphabet, ψ and θ two OPTL formulae on it, and $\mathcal{A}_{\psi\, \mathcal{U}^{\downarrow}\, \theta}$ an OPA satisfying rules 1-4 of Section 5.7. For any word $w = \#xyz\#$ on $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$, let position $j = |xy|$ in w: we have*

$$(w, j) \models \psi\, \mathcal{U}^{\downarrow}\, \theta$$

if and only if $\mathcal{A}_{\psi\, \mathcal{U}^{\downarrow}\, \theta}$ performs one computation that brings it from configuration $\langle yz, \Phi, \alpha\gamma\rangle$ to a configuration $\langle z, \Phi', \alpha'\gamma\rangle$ such that $|\alpha| = 1$, $|\alpha'| \geq 1$, $\psi\, \mathcal{U}^{\downarrow}_s\, \theta$, $\psi\, \mathcal{U}^{\downarrow}_{end}\, \theta \notin \Phi'$, and $\psi\, \mathcal{U}^{\downarrow}\, \theta \in \Phi_j$, where Φ_j is the state of the automaton before reading position j.

Moreover, if $(w, j) \models \psi\, \mathcal{U}^{\downarrow}\, \theta$, then $\psi\, \mathcal{U}^{\downarrow}\, \theta \in \Phi_j$ in all accepting runs of $\mathcal{A}_{\psi\, \mathcal{U}^{\downarrow}\, \theta}$

Proof. [⇒] Let $d \in \mathscr{P}(AP)$ be the label of position j. If $\psi\, \mathcal{U}^{\downarrow}\, \theta$ holds in j, then it is the right context of at least two chains, so w is of the form of Fig. 15.

Also, there exists one value $1 \leq k \leq n$ such that $(w, i_{b_k}) \models \theta$ and for all $k < p \leq n$, $(w, i_{b_p}) \models \psi \wedge \neg\theta$. The automaton's computation on w proceeds normally until it reaches b_n, in configuration $\langle b_n \ldots z, \Phi_{b_n}, \sigma\rangle$, where $\sigma = [c^n_{m_0}, \Phi_{c^n_0}]\alpha\gamma$, or $\sigma = \alpha\gamma$, $\alpha = [a, \Phi_a]$ being the stack symbol introduced or updated when reading a. By hypothesis, either $\theta \in \Phi_{b_n}$, or $\theta \notin \Phi_{b_n}$ and $\psi \in \Phi_{b_n}$. In both cases, the computation proceeds normally, until it reaches c^{n-1}_0 (or b_{n-1}, if c^{n-1}_0 does not exist), with configuration $\langle c^{n-1}_0 \ldots z, \Phi_{c^{n-1}_0}, \sigma\rangle$. By construction, $\ominus_\chi\theta \vee (\neg \ominus_\chi \top \wedge \ominus\theta) \in \Phi_{c^{n-1}_0}$ (resp. $\ominus_\chi\psi \vee (\neg \ominus_\chi \top \wedge \ominus\psi) \in \Phi_{c^{n-1}_0}$). This holds by correctness of model checking for the $\ominus_\chi$ and $\ominus$ operators. In particular, if $v^{n-1}_0 \neq \varepsilon$, then $\ominus_\chi\theta$ (resp. $\ominus_\chi\psi$) holds, and $\neg \ominus_\chi \top \wedge \ominus\theta$ (resp. $\neg \ominus_\chi \top \wedge \ominus\psi$) holds otherwise. The computation goes on in the same way for each b_p, with $k \leq p \leq n$. Finally, right after reading $c^0_{m_0}$, the automaton is in configuration $\langle dz, \Phi_d, [c^0_{m_0}, \Phi_{c^0_0}][b_1, \Phi_{c^1_0}] \ldots [b_n, \Phi_{c^n_0}]\alpha\gamma\rangle$, with $\psi\, \mathcal{U}^{\downarrow}_s\, \theta$, $\psi\, \mathcal{U}^{\uparrow}_{end}\, \theta \notin \Phi_d$, $\ominus_\chi\theta \vee (\neg \ominus_\chi \top \wedge \ominus\theta) \in \Phi_{c^{k-1}_0}$ and $\ominus_\chi\psi \vee (\neg \ominus_\chi \top \wedge \ominus\psi) \in \Phi_{c^{p-1}_0}$ for $k < p \leq n$. The OPA guesses that $\psi\, \mathcal{U}^{\downarrow}\, \theta$ holds in d, so $\psi\, \mathcal{U}^{\downarrow}\, \theta \in \Phi_d$. Now, since $c^0_{m_0}$ and all $b_1 \ldots b_n$ take precedence from d, a series of pop transitions occur. When popping $[b_{k-1}, \Phi_{c^{k-1}_0}]$, $\psi\, \mathcal{U}^{\downarrow}_s\, \theta$ is introduced in the resulting state by rule 4, and propagated in all transitions until configuration $\langle dz, \Phi^{(n-1)}_d, [b_n, \Phi_{c^n_0}]\alpha\gamma\rangle$, with $\psi\, \mathcal{U}^{\downarrow}_s\, \theta \in \Phi^{(n-1)}_d$. Then, by rule 3, the last pop move leads to $\langle dz, \Phi^{(n)}_d, \alpha\gamma\rangle$,

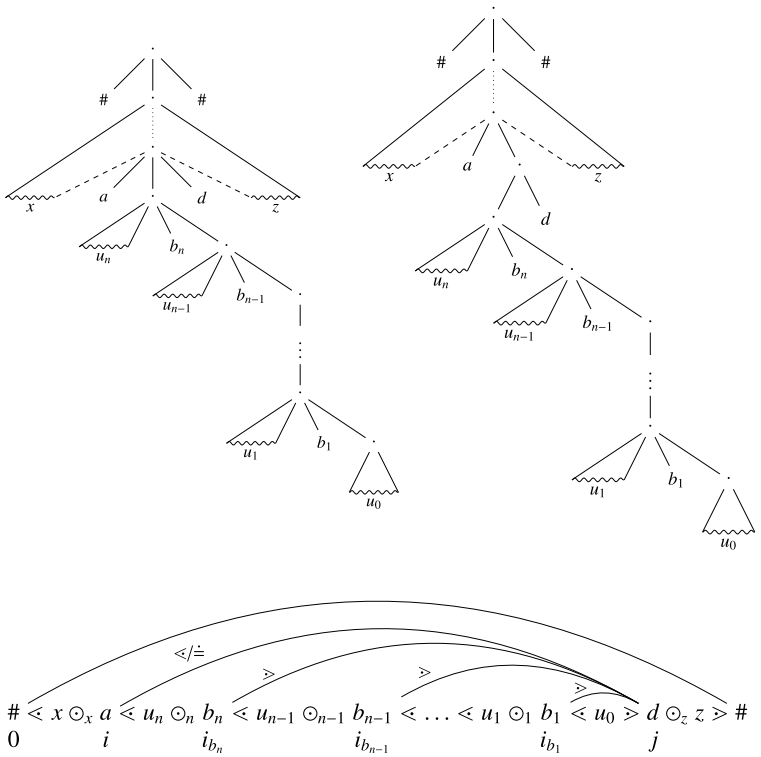


Fig. 15. The two possible ASTs of a generic OP word $w = xyz$ (top) expanded on the rightmost non-terminal, and its flat representation with chains (bottom). Wavy lines are placeholders for subtree frontiers. We have either $a \doteq d$ (top left) or $a \lessdot d$ (top right), and $b_k \gtrdot d$ for $1 \leq k \leq n$. For $1 \leq k \leq n$, we either have ${}^{b_{k+1}}[u_k]^{b_k}$, or u_k is of the form $v_0^k c_0^k v_1^k c_1^k \ldots c_{m_k}^k v_{m_k+1}^k$, where $c_p^k \doteq c_{p+1}^k$ for $0 \leq p < m_k$, $c_{m_k}^k \doteq b_k$, and resp. $a \lessdot c_0^n$ and $b_{k+1} \lessdot c_0^k$. Moreover, for each $0 \leq p < m_k$, either $v_{p+1}^k = \varepsilon$ or ${}^{c_p^k}[v_{p+1}^k]^{c_{p+1}^k}$; either $v_{m_k+1}^k = \varepsilon$ or ${}^{c_{m_k}^k}[v_{m_k+1}^k]^{b_k}$, and either $v_0^k = \varepsilon$ or ${}^{b_{k+1}}[v_0^k]^{c_0^k}$ (resp. ${}^{a}[v_0^n]^{c_0^n}$). u_0 has the same form, except $v_{m_0}^0 = \varepsilon$ and $c_{m_0}^0 \gtrdot d$. The $\odot$ symbols are placeholders for precedence relations, and they vary depending on the surrounding terminal characters.

with $\psi\, \mathcal{U}_{end}^{\downarrow}\, \theta \in \Phi_d^{(n)}$. It can then continue by reading d with a push (leaving α untouched) or a shift move (updating α), which verifies the previous guess, according to rule 2.

Note that, by rule 3, $\psi\, \mathcal{U}_{end}^{\downarrow}\, \theta$ is propagated together with $\psi\, \mathcal{U}_s^{\downarrow}\, \theta$ in pop moves that identify a valid take-precedence hierarchical path. Thus, by rule 2, $\psi\, \mathcal{U}^{\downarrow}\, \theta$ appears in Φ_j in all computations in which $\psi\, \mathcal{U}^{\downarrow}\, \theta$ holds in j.

[$\Leftarrow$] Observe that d must be the right context of at least two chains. By rule 2, the state Φ_d visited by the automaton right before reading d with a shift or a push move contains $\psi\, \mathcal{U}_{end}^{\downarrow}\, \theta$. If, by contradiction, no chain ended in d, then the previous transition would have been a shift or a push. By rule 1 no such transition exists. If only one chain ended in d, the previous transition would have been a pop, starting from a state Φ'_d with $\psi\, \mathcal{U}_s^{\downarrow}\, \theta \in \Phi'_d$ by rule 3. But this would be the state resulting from the shift/push move reading the symbol before d (i.e. $c_{m_0}^0$), which would violate rule 1. Hence, w must be of the form of Fig. 15, or any computation on it would stop before reading d.

For this side of the implication, the proof is analogous to the other one. It amounts to noting that a computation resulting in a push/shift move reading d must contain a series of pop transitions such as the one described above, by rules 2, 3 and 4. Rule 4 constrains $\ominus_\chi \psi \vee (\neg \ominus_\chi \top \wedge \ominus \psi)$ or $\ominus_\chi \theta \vee (\neg \ominus_\chi \top \wedge \ominus \theta)$ to hold in the appropriate positions, and by correctness of $\ominus_\chi$ and $\ominus$, this results in the existence of a value $1 \leq k \leq n$ such that $\theta \in \Phi_{b_k}$ and for each $k < p \leq n$, $\psi \in \Phi_{b_p}$. Such a k must exist, or $\psi\, \mathcal{U}_s^{\downarrow}\, \theta$ would be propagated backwards in the final sequence of pop moves according to rule 4, down to the state resulting from the push/shift transition reading $c_{m_0}^0$, the symbol before d. But such a transition would violate rule 1. □

5.8. Take-precedence hierarchical since ($\mathcal{S}^{\uparrow}$) operator

Let $\Phi \in \text{Atoms}(\varphi)$ be a state in which $\psi\, \mathcal{S}^{\uparrow}\, \theta$ holds. Then we add $\psi, \theta \in \text{Cl}(\varphi)$, $\psi\, \mathcal{S}_s^{\uparrow}\, \theta, \psi\, \mathcal{S}_{end}^{\uparrow}\, \theta \in \text{Cl}_{aux}(\varphi)$, and we impose the following constraints on the transition function. For any $\Phi, \Theta \in \text{Atoms}(\varphi)$, and $a \in \mathscr{P}(AP)$, let $(\Phi, a, \Theta) \in \delta_{push} \cup \delta_{shift}$: then

1. $\psi\, \mathcal{S}^{\uparrow}\, \theta \in \Theta$, iff $\psi\, \mathcal{S}_s^{\uparrow}\, \theta \in \Theta$;
2. if $\psi\, \mathcal{S}^{\uparrow}\, \theta \in \Phi$, then $\psi\, \mathcal{S}_{end}^{\uparrow}\, \theta \in \Phi$ and $\psi\, \mathcal{S}_s^{\uparrow}\, \theta \notin \Phi$.

For any $\Phi, \Theta, \Psi \in \text{Atoms}(\varphi)$, let $(\Phi, \Theta, \Psi) \in \delta_{pop}$: we have

3. if $\psi\ \mathcal{S}^{\uparrow}\ \theta \in \Psi$ then $\psi\ \mathcal{S}^{\uparrow}\ \theta \in \Phi$;
4. $\psi\ \mathcal{S}_s^{\uparrow}\ \theta \in \Phi$ iff, either, $(\ominus_\chi \theta \vee (\neg \ominus_\chi \top \wedge \ominus\theta) \in \Theta$ and $\psi\ \mathcal{S}_{end}^{\uparrow}\ \theta \notin \Psi)$, or $(\ominus_\chi \psi \vee (\neg \ominus_\chi \top \wedge \ominus\psi) \in \Theta$ and $\psi\ \mathcal{S}_s^{\uparrow}\ \theta \in \Psi)$;
5. $\psi\ \mathcal{S}_{end}^{\uparrow}\ \theta \notin \Phi$.

Finally, if $\psi\ \mathcal{S}^{\uparrow}\ \theta \in \Phi$, then $\Phi \notin I$.

The constraints for this operator work in a way similar to those of $\mathcal{U}^{\downarrow}$. Here, the auxiliary operator $\mathcal{S}_{end}^{\uparrow}$ is used to mark the state of the automaton just before reading the right context of the chains, in which $\psi\ \mathcal{S}^{\uparrow}\ \theta$ holds, and discern the pop move related to the end of the backward-maximal chain. Its left context cannot be part of the take-precedence hierarchical path, and the presence of $\mathcal{S}_{end}^{\uparrow}$, with rule 4, prevents it from ending there. Moreover, constraint 3 forces $\psi\ \mathcal{S}^{\uparrow}\ \theta$ to be in the current state since the first *pop* transition, so that 1 always puts $\psi\ \mathcal{S}_s^{\uparrow}\ \theta$ into the state of the OPA before such transition.

We prove the correctness of these constraints in the following lemma.

Lemma 5.8. *Let AP be a finite set of atomic propositions, $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$ an OP alphabet, ψ and θ two OPTL formulae on it, and $\mathcal{A}_{\psi\mathcal{S}^{\uparrow}\theta}$ an OPA satisfying rules 1-5 of Section 5.8. For any word $w = \#xyz\#$ on $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$, let position $j = |xy|$ in w: we have*

$$(w, j) \models \psi\ \mathcal{S}^{\uparrow}\ \theta$$

if and only if $\mathcal{A}_{\psi\mathcal{S}^{\uparrow}\theta}$ performs one computation that brings it from configuration $\langle yz, \Phi, \alpha\gamma\rangle$ to a configuration $\langle z, \Phi', \alpha'\gamma\rangle$ such that $|\alpha| = 1$, $|\alpha'| \geq 1$, $\psi\ \mathcal{S}_s^{\uparrow}\ \theta$, $\psi\ \mathcal{S}_{end}^{\uparrow}\ \theta \notin \Phi'$, and $\psi\ \mathcal{S}^{\uparrow}\ \theta \in \Phi_j$, where Φ_j is the state of the automaton before reading position j.

Moreover, if $(w, j) \models \psi\ \mathcal{S}^{\uparrow}\ \theta$, then $\psi\ \mathcal{S}^{\uparrow}\ \theta \in \Phi_j$ in all accepting runs of $\mathcal{A}_{\psi\mathcal{S}^{\uparrow}\theta}$.

Proof. [$\Rightarrow$] By the semantics of $\psi\ \mathcal{S}^{\uparrow}\ \theta$, j is the right context of at least two chains, so w has the form of Fig. 15. If $\psi\ \mathcal{S}^{\uparrow}\ \theta$ holds in j, then there exists a value $1 \leq k \leq n$ such that $(w, i_{b_k}) \models \theta$ and for each $1 \leq p < k$, $(w, i_{b_p}) \models \psi$. The behavior of the automaton on such a word is essentially analogous to the one described in the proof of Lemma 5.7. Therefore, the OPA reads the first part of w, reaching the configuration $\langle dz, \Phi_d, [c^0_{m_0}, \Phi_{c^0_0}][b_1, \Phi_{c^1_0}] \dots [b_n, \Phi_{c^n_0}]\alpha\gamma\rangle$, where $\alpha\gamma$ is the stack content right after reading a, $\ominus_\chi\theta \vee (\neg \ominus_\chi \top \wedge \ominus\theta) \in \Phi_{c^{k-1}_0}$ because $\theta \in \Phi_{b_k}$, and for all $1 \leq p < k$, $\ominus_\chi\psi \vee (\neg \ominus_\chi \top \wedge \ominus\psi) \in \Phi_{c^{p-1}_0}$, because $\psi \in \Phi_{b_p}$. The OPA guesses that $\psi\ \mathcal{S}^{\uparrow}\ \theta$ holds in j, so $\psi\ \mathcal{S}^{\uparrow}\ \theta \in \Phi_d$, and by rule 1, $\psi\ \mathcal{S}_s^{\uparrow}\ \theta \in \Phi_d$. Then, a series of pop moves occur. By rule 4, all transitions occurring before the one popping $[b_{k-1}, \Phi_{c^{k-1}_0}]$ propagate $\psi\ \mathcal{S}_s^{\uparrow}\ \theta$ into the next state. Then, $[b_{k-1}, \Phi_{c^{k-1}_0}]$ is popped, leading to $\langle dz, \Phi^{(k)}_d, [b_k, \Phi_{c^k_0}] \dots [b_n, \Phi_{c^n_0}]\alpha\gamma\rangle$. This move may occur, verifying the guess, due to rule 4, because $\psi\ \mathcal{S}_{end}^{\uparrow}\ \theta \notin \Phi^{(k)}_d$. The subsequent move is, in fact, the pop of $[b_k, \Phi_{c^k_0}]$, so rule 5 applies. Thus, the run may proceed with a push or a shift move reading d, satisfying rule 2. If d is read with a push, then α is left untouched, and it is only updated if d is read with a shift move.

Note that the propagation of $\psi\ \mathcal{S}_s^{\uparrow}\ \theta$ in pop transitions occurs whenever there is a valid path satisfying $\psi\ \mathcal{S}^{\uparrow}\ \theta$. So, by rule 1 also appears, and is kept in the current state until d is read, so $\psi\ \mathcal{S}^{\uparrow}\ \theta \in \Phi_j$.

[$\Leftarrow$] For the other side of the implication, if j is the right context of no chain, the push/shift move reading position j is preceded by another push/shift move. By rule 1, the latter introduces $\psi\ \mathcal{S}_s^{\uparrow}\ \theta$ in the next state Φ_j, which prevents the former move from taking place, by rule 2. If j is the right context of only one chain, then $\psi\ \mathcal{S}_s^{\uparrow}\ \theta$ is introduced by rule 1 into the starting state of the pop move related to the end of that chain. Due to rule 4, either $\psi\ \mathcal{S}_s^{\uparrow}\ \theta \in \Phi_j$ or $\psi\ \mathcal{S}_{end}^{\uparrow}\ \theta \notin \Phi_j$, preventing the transition that would read position j, by rule 2. So, w must be of the form of Fig. 15.

Now, we must prove that any successful computation must be of the form we described earlier. The fact that $\psi\ \mathcal{S}_s^{\uparrow}\ \theta$ is contained in state Φ_d is enforced by rule 1. Note that, due to rule 3, $\psi\ \mathcal{S}^{\uparrow}\ \theta$ must already be present in Φ_d, and it cannot be introduced in one of the subsequent pop transitions. Hence, for the push/shift move reading d not to violate rule 2, $\psi\ \mathcal{S}_s^{\uparrow}\ \theta$ must stop being propagated by rule 4. This can only happen if there exists $1 \leq k \leq n$ such that $\ominus_\chi\psi \vee (\neg \ominus_\chi \top \wedge \ominus\psi)$ holds in all popped states until $\Phi_{c^{k-1}_0}$, which contains $\ominus_\chi\theta \vee (\neg \ominus_\chi \top \wedge \ominus\theta)$. Then, we have $\theta \in \Phi_{b_k}$ and $\psi \in \Phi_{b_p}$, for all $1 \leq p < k$, where Φ_{b_h} is the state of the automaton before reading b_h, for any h. □

5.9. Global correctness

The correctness of this model checking procedure is carried out by induction on the syntactic structure of the formula. For each operator, we prove that if the automaton built for its operands is correct, then the one built for the whole formula is also correct.

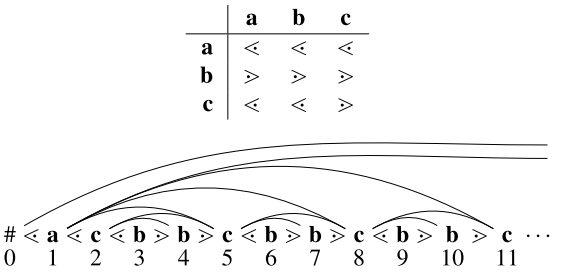


Fig. 16. OPM (top) and the structure of ω-word $\mathbf{a}(\mathbf{cbb})^\omega$ (bottom).

Theorem 5.9. *Given a finite set of atomic propositions AP, an OP alphabet $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$, a word w on it, and an OPTL formula φ, the automaton built according to the procedure in this section is such that we have*

$$(w, 1) \models \varphi$$

if and only if it performs at least one accepting computation on w. Moreover, such an automaton is of size $2^{O(|\varphi|)}$.

Proof. In Lemmas 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, and 5.8 we proved that any OPTL operator holds in a word position iff the corresponding automaton performs at least one computation that, after reading a (possibly improper) prefix of the word, brings it in a state containing no auxiliary operators. Moreover, each operator appears in an OPA state only if it holds in the corresponding word position, or the computation is not accepting. We take the correctness of LTL operators for granted.

[$\Rightarrow$] If φ holds in position 1, then all temporal requirements of φ and its subformulas are satisfied before the end of w. Therefore, by the lemmas listed above, there exists a computation of the OPA built for φ and a prefix x of w such that, after reading x, the current state of the automaton does not contain any auxiliary symbol or temporal operator related to a subformula of φ. Then, by the fact that all consistent states and transitions are part of the automaton, there exists at least one computation that reads the rest of w (if any) ending in a state containing no future temporal operators, which is final. Such a computation is accepting.

[$\Leftarrow$] Suppose the automaton built for φ accepts the word w. Then there exists an accepting computation starting from the initial state containing φ and ending in a final state. By construction, if a state is final it contains no auxiliary or future temporal operators. Therefore, it follows by the lemmas listed above that φ holds in position 1.

Regarding the complexity claim, the size of the set $\mathrm{Cl}(\varphi)$ is linear w.r.t. the length of formula φ, and its increase due to the addition of auxiliary operators in the subsequent sections is also linear. The set $\mathrm{Atoms}(\varphi)$, which contains the states of the automaton, is a subset of $\mathscr{P}(\mathrm{Cl}(\varphi))$; thus, its size is at most exponential in $\mathrm{Cl}(\varphi)$ and in $|\varphi|$. □

Given a formula φ it is possible to check its satisfiability by building OPA $\mathcal{A}_\varphi$ of size $2^{O(|\varphi|)}$, and then checking its emptiness. This can be done in time polynomial in the size of $\mathcal{A}_\varphi$ by transforming $\mathcal{A}_\varphi$ into an equivalent context-free grammar [27], and testing emptiness of the generated language [38]. Moreover, EXPTIME-hardness can be deduced from the same result as for NWTL [12] and Theorem 4.5. Thus, we claim

Corollary 5.10. *The satisfiability problem for OPTL on finite words is EXPTIME-complete.*

6. OPTL on infinite words and its model checking

In the previous sections, we studied OPTL with semantics based on finite words. In order to extend its semantics to the infinite case, it suffices to consider an infinite set of positions $U = \mathbb{N}$. The formal definitions of all OPTL operators remain the same as described in Section 3.2.

As for the intuitive meaning, the only change concerns "forward-maximal chains" ($\overrightarrow{\chi}$). We formally defined $\overrightarrow{\chi}(i, j)$ to hold if $\chi(i, j)$ and $i \doteq j$ or $i \gtrdot j$. In finite OP words, one of such chains is also the outermost one starting in a position i. In ω-words, the outermost chain may be *open* (cf. Section 2.1). In this case, according to the formal semantics of Section 3.2, $\overrightarrow{\chi}(i, j)$ does not hold for any j in the context of an open chain. E.g., in the example of Fig. 16, there is no j such that $\overrightarrow{\chi}(1, j)$ holds, and $\bigcirc_\chi \top$ is false in 1.

The relations with nested words stated in Section 4 extend naturally when shifting to ω-words. Instead, the model checking procedure for the finite words case becomes slightly more complex for ω-words, due to the need to adapt it to Büchi acceptance conditions.

Model checking for an OPTL formula φ on infinite words can be performed by building a generalized ωOPBA (cf. Section 2.1) $\mathcal{A}^\omega_\varphi = \langle \mathscr{P}(AP), M_{\mathscr{P}(AP)}, \mathrm{Atoms}(\varphi) \times \mathrm{Cl}_{stack}(\varphi), I, \mathbf{F}, \delta \rangle$, which differs from the finite-word counterpart only for the state set and the acceptance condition. As in [12], the generalized Büchi acceptance condition is a slight variation on the one shown in Section 2.1: $\mathbf{F}$ 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 **F** is visited infinitely often during the computation. 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 considering states of the form $\Phi = (\Phi^c, \Phi^p)$, $\Phi^c \in \text{Atoms}(\varphi)$ and $\Phi^p \in \text{Cl}_{stack}(\varphi)$. Φ^c is used exactly in the same way as states in the finite counterpart. Instead, Φ^p, called the *pending* part of Φ, contains only auxiliary operators in $\text{Cl}_{stack}(\varphi) \subseteq \text{Cl}_{aux}(\varphi)$ that are in a stack symbol currently on the stack. Thus, pending temporal requirements are moved from the stack into the OPA state, and they can be considered in the Büchi acceptance condition.

Suppose we want to model check $\bigcirc_\chi \psi$. The auxiliary symbol $\bigcirc^s_\chi \psi$ must be inserted in the pending part of the current state whenever a stack symbol containing it is pushed, and kept in the automaton's state until that symbol is popped, and its temporal requirement satisfied. Then, it is possible to define an acceptance set $F \in \mathbf{F}$, for each use of this temporal operator, as the set of states not containing $\bigcirc^s_\chi \psi$ in their atom, nor in their pending part.

This construction is formalized as follows. Let $\psi \in \text{Cl}_{stack}(\varphi)$. We add a few constraints on the transition relations. For any $\Phi, \Theta, \Psi \in \text{Atoms}(\varphi)$ and $a \in \mathscr{P}(AP)$, let $(\Phi, a, \Theta) \in \delta_{push}$:

1. if $\psi \in \Phi^c$, then $\psi \in \Theta^p$;

let $(\Phi, a, \Theta) \in \delta_{push}$ or $(\Phi, a, \Theta) \in \delta_{shift}$:

2. if $\psi \in \Phi^p$, then $\psi \in \Theta^p$;

let $(\Phi, \Theta, \Psi) \in \delta_{pop}$:

3. if $\psi \in \Phi^p$ and $\psi \in \Theta^p$, then $\psi \in \Psi^p$.

Lemma 6.1. *Let AP be a finite set of atomic propositions, $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$ an OP alphabet, $\psi \in \text{Cl}_{stack}(\varphi)$, and $\mathcal{A}$ an OPA satisfying rules 1-3 above. For any ω-word $w = \#xy$ on $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$, let $\langle y, \Phi, \gamma \rangle$ be $\mathcal{A}$'s configuration after reading x.*

If there exists a stack symbol $[a, \Theta] \in \gamma$ such that $\psi \in \Theta^c$, then $\psi \in \Phi^p$.

Proof. Suppose there exists a stack symbol $[a, \Theta] \in \gamma$ such that $\psi \in \Theta^c$, and $\psi \notin \Theta^p$. This symbol was put on the stack by a previous push move. Let $\langle azy, \Theta, \delta \rangle$, where az is a suffix of x, be the configuration of the OPA just before such move.

By rule 1, this push move brings the OPA to a configuration $\langle zy, \Phi_z, [a, \Theta]\delta \rangle$, with $\psi \in \Phi^p_z$. By rule 2, ψ is propagated in the pending part of $\mathcal{A}$'s state by all subsequent push and shift transitions. Consequently, all stack symbols pushed on top of $[a, \Theta]$ contain ψ in their pending part. Thus, according to rule 3, ψ is also propagated every time one of such symbols is popped. When $[a, \Theta]$ is popped, rule 3 does not apply, so ψ may be dropped from the next state.

Conversely, suppose $\psi \in \Theta^p$. In this case, the above reasoning also applies. The only difference is that rule 3 does not apply on the transition popping $[a, \Theta]$, so ψ continues being propagated afterwards.

Thus, whenever $[a, \Theta]$ is present in the stack, ψ is in the pending part of the current state. It is dropped only once $[a, \Theta]$ is popped. □

All future operators except $\mathcal{S}^\downarrow$ only need additions to the construction for the finite case (Section 5), which we detail in the following sections. Since the construction for $\mathcal{S}^\downarrow$ on finite words relies on the $\bigcirc_\chi$ operator, it does not need to be modified. All constraints on the transition relation defined in Section 5 must be considered valid for the atom of ωOPBA states they refer to. We omit the construction for LTL operators, because we resort to the classical one. Past operators do not need any addition to the rules for the finite case, because infinite words pose no further issues in ensuring their eventual satisfaction.

After separately proving the correctness of the construction for each operator, we inductively prove the soundness of the whole construction in Theorem 6.5, as we did for the finite case.

6.1. Matching next ($\bigcirc_\chi$) operator

In order to adapt the OPA construction of Section 5.2, it suffices to add $\bigcirc^s_\chi \psi \in \text{Cl}_{stack}(\varphi)$ if $\bigcirc_\chi \psi \in \text{Cl}(\varphi)$, so that its presence in the stack is tracked by the pending part of the OPA's states. Moreover, we define $F_{\bigcirc_\chi \psi} \in \mathbf{F}$, the Büchi acceptance set for $\bigcirc_\chi \psi$, so that for any $\Phi \in \text{Atoms}(\varphi) \times \text{Cl}_{stack}(\varphi)$, $\Phi \in F_{\bigcirc_\chi \psi}$ iff $\bigcirc^s_\chi \psi \notin \Phi^c$ and $\bigcirc^s_\chi \psi \notin \Phi^p$.

We show how the construction works with the help of the example of Fig. 17. The top part shows an accepting run of the ωOPBA for $\bigcirc_\chi \mathbf{c}$. It proceeds in the same way as shown in Section 5.2, except the word is infinite, and $\bigcirc_\chi \mathbf{c}$ is satisfied in every **c**. The ωOPBA guesses this fact by visiting a state containing $\bigcirc_\chi \mathbf{c}$ before reading each **c**, starting from the initial state q_0 in step 1, then in step 5, and so on. $\bigcirc^s_\chi \mathbf{c}$ is inserted into the atom of state q_1 when $\bigcirc_\chi \mathbf{c}$ appears due to rule F 1 of the finite-word construction (step 1-2). When q_1 is pushed on the stack, $\bigcirc^s_\chi \mathbf{c}$ enters the pending part of the next state (step 2-3), due to rule ω 1 of Section 6. When q_1 is popped, rule ω 3 does not apply, and the ωOPBA guesses not to propagate

	input	state	stack	rel.	move	rule
1	$\mathbf{c} \lessdot \mathbf{b} \gtrdot \mathbf{b} \gtrdot \mathbf{c} \lessdot \mathbf{b} \cdots$	$q_0 = (\{\mathbf{c}, \bigcirc_\chi \mathbf{c}\}, \emptyset)$	$\bot$	$\# \lessdot \mathbf{c}$	push	F1, F2
2	$\mathbf{b} \gtrdot \mathbf{b} \gtrdot \mathbf{c} \lessdot \mathbf{b} \cdots$	$q_1 = (\{\mathbf{b}, \bigcirc^s_\chi \mathbf{c}\}, \emptyset)$	$[\mathbf{c}, q_0]\bot$	$\mathbf{c} \lessdot \mathbf{b}$	push	ω1
3	$\mathbf{b} \gtrdot \mathbf{c} \lessdot \mathbf{b} \cdots$	$(\{\mathbf{b}\}, \{\bigcirc^s_\chi \mathbf{c}\})$	$[\mathbf{b}, q_1][\mathbf{c}, q_0]\bot$	$\mathbf{b} \gtrdot \mathbf{b}$	pop	F5
4	$\mathbf{b} \gtrdot \mathbf{c} \lessdot \mathbf{b} \cdots$	q_1	$[\mathbf{c}, q_0]\bot$	$\mathbf{c} \lessdot \mathbf{b}$	push	ω1
5	$\mathbf{c} \lessdot \mathbf{b} \cdots$	$(\{\mathbf{c}, \bigcirc_\chi \mathbf{c}\}, \{\bigcirc^s_\chi \mathbf{c}\})$	$[\mathbf{b}, q_1][\mathbf{c}, q_0]\bot$	$\mathbf{b} \gtrdot \mathbf{c}$	pop	F5
6	$\mathbf{c} \lessdot \mathbf{b} \cdots$	$(\{\mathbf{c}, \bigcirc_\chi \mathbf{c}, \bigcirc^{end}_\chi \mathbf{c}\}, \emptyset)$	$[\mathbf{c}, q_0]\bot$	$\mathbf{c} \gtrdot \mathbf{c}$	pop	F3, F4
7	$\mathbf{c} \lessdot \mathbf{b} \cdots$	q_0	$\bot$	$\# \lessdot \mathbf{c}$	push	F1, F2

	input	state	stack	rel.	move	rule
1	$\mathbf{a} \lessdot \mathbf{c} \lessdot \mathbf{b} \gtrdot \mathbf{b} \gtrdot \mathbf{c} \lessdot \mathbf{b} \cdots$	$q_2 = (\{\mathbf{a}, \bigcirc_\chi \mathbf{c}\}, \emptyset)$	$\bot$	$\# \lessdot \mathbf{a}$	push	F1,F2
2	$\mathbf{c} \lessdot \mathbf{b} \gtrdot \mathbf{b} \gtrdot \mathbf{c} \lessdot \mathbf{b} \cdots$	$q_3 = (\{\mathbf{c}, \bigcirc_\chi \mathbf{c}, \bigcirc^s_\chi \mathbf{c}\}, \emptyset)$	$[\mathbf{a}, q_2]\bot$	$\mathbf{a} \lessdot \mathbf{c}$	push	F1,F2
3	$\mathbf{b} \gtrdot \mathbf{b} \gtrdot \mathbf{c} \lessdot \mathbf{b} \cdots$	$q_4 = (\{\mathbf{b}, \bigcirc^s_\chi \mathbf{c}\}, \{\bigcirc^s_\chi \mathbf{c}\})$	$[\mathbf{c}, q_3][\mathbf{a}, q_2]\bot$	$\mathbf{c} \lessdot \mathbf{b}$	push	ω1
4	$\mathbf{b} \gtrdot \mathbf{c} \lessdot \mathbf{b} \cdots$	$(\{\mathbf{b}\}, \{\bigcirc^s_\chi \mathbf{c}\})$	$[\mathbf{b}, q_4][\mathbf{c}, q_3][\mathbf{a}, q_2]\bot$	$\mathbf{b} \gtrdot \mathbf{b}$	pop	F5,ω3
5	$\mathbf{b} \gtrdot \mathbf{c} \lessdot \mathbf{b} \cdots$	q_4	$[\mathbf{c}, q_3][\mathbf{a}, q_2]\bot$	$\mathbf{c} \lessdot \mathbf{b}$	push	ω1
6	$\mathbf{c} \lessdot \mathbf{b} \cdots$	$q_5 = (\{\mathbf{c}, \bigcirc_\chi \mathbf{c}\}, \{\bigcirc^s_\chi \mathbf{c}\})$	$[\mathbf{b}, q_4][\mathbf{c}, q_3][\mathbf{a}, q_2]\bot$	$\mathbf{b} \gtrdot \mathbf{c}$	pop	F5,ω3
7	$\mathbf{c} \lessdot \mathbf{b} \cdots$	$(\{\mathbf{c}, \bigcirc_\chi \mathbf{c}, \bigcirc^{end}_\chi \mathbf{c}\}, \{\bigcirc^s_\chi \mathbf{c}\})$	$[\mathbf{c}, q_3][\mathbf{a}, q_2]\bot$	$\mathbf{c} \gtrdot \mathbf{c}$	pop	F3,F4,F5
8	$\mathbf{c} \lessdot \mathbf{b} \cdots$	q_3	$[\mathbf{a}, q_2]\bot$	$\mathbf{a} \lessdot \mathbf{c}$	push	F1,F2

Fig. 17. An accepting run of the ωOPBA for formula $\bigcirc_\chi \mathbf{c}$ on the ω-word $(\mathbf{cbb})^\omega$ (top) w.r.t. the OPM of Fig. 16, and a rejecting one on $\mathbf{a}(\mathbf{cbb})^\omega$ (bottom). Rules preceded by 'F' are from Section 5.2, those with 'ω' from Section 6.

$\bigcirc^s_\chi \mathbf{c}$ in the pending part of the next state. Notice how $\bigcirc^s_\chi \mathbf{c}$ remains in one of the two parts of the current state until step 6, in which $\bigcirc_\chi \mathbf{c}$ of step 1 is satisfied. The ωOPBA guesses its satisfaction, removing $\bigcirc^s_\chi \mathbf{c}$ from the state of step 7, and comes back to the same configuration of step 1. Since this subword is repeated indefinitely, state q_0, which is final, is also visited infinitely often, causing the acceptance of the word.

The bottom part of Fig. 17 shows a rejecting run. It starts in a state containing $\bigcirc_\chi \mathbf{c}$ as all initial states of this ωOPB do, but **a** is the left context of an open chain, so $\bigcirc_\chi \mathbf{c}$ is not satisfied. $\bigcirc^s_\chi \mathbf{c}$ starts being propagated in step 2, and it remains in the current state forever. Notice that, every time a pop occurs, rule F 5 or ω 3 applies, and $\bigcirc^s_\chi \mathbf{c}$ is never dropped. So, no final state is visited infinitely often.

We show the correctness of this construction below.

Lemma 6.2. *Let AP be a finite set of atomic propositions, $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$ an OP alphabet, ψ an OPTL formula on it, and $\mathcal{A}_{\bigcirc_\chi \psi}$ an ωOPBA satisfying rules 1-5 of Section 5.2, and 1-3 of Section 6. For any ω-word $w = \#xyz$ on $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$, let position $i = |x| + 1$ in w: we have*

$$(w, i) \models \bigcirc_\chi \psi$$

if and only if the automaton built for $\bigcirc_\chi \psi$ performs at least one computation such that either:

a) the automaton goes through a configuration $\langle yz, \Phi, \rho \rangle$, with $\bigcirc_\chi \psi \in \Phi^c$ and $\bigcirc^s_\chi \psi \notin \Phi^p$, and the computation is accepting; or

b) the automaton passes from configuration $\langle yz, \Phi_0, \alpha\gamma \rangle$ with $\bigcirc_\chi \psi \in \Phi^c_0$ and $\bigcirc^s_\chi \psi \in \Phi^p_0$ to $\langle z, \Phi_n, \alpha'\gamma \rangle$ such that $\bigcirc^s_\chi \psi \notin \Phi^c_n$, $|\alpha| = 1$, $|\alpha'| \leq 1$, and for all $1 \leq k \leq n$, $\bigcirc^s_\chi \psi \in \Phi^c_k$ or $\bigcirc^s_\chi \psi \in \Phi^p_k$.

Proof. [$\Rightarrow$] If $(w, i) \models \bigcirc_\chi \psi$, by Lemma 5.2 the OPA performs at least one computation that brings it from configuration $C_0 = \langle yz, \Phi, \alpha\gamma \rangle$ with $\bigcirc_\chi \psi \in \Phi^c$ to $C_f = \langle z, \Phi', \alpha'\gamma \rangle$, such that $\bigcirc^s_\chi \psi, \bigcirc^{end}_\chi \psi \notin \Phi'^c$, $|\alpha| = 1$ and $|\alpha'| \leq 1$.

Suppose $\bigcirc^s_\chi \psi \notin \Phi^p$ (case a)). The proof of Lemma 5.2 shows that, when $|\alpha'| = 1$, the state contained in it is the same one contained in α. If $\bigcirc^s_\chi \psi \notin \Phi^p$, then $\bigcirc^s_\chi \psi$ is not present in any state in $\alpha\gamma$, nor $\alpha'\gamma$. If any state containing $\bigcirc^s_\chi \psi$ in its atom is pushed between C_0 and C_f, it is popped before reaching C_f. Therefore, by Lemma 6.1 we can conclude that there exists a computation in which $\bigcirc^s_\chi \psi \notin \Phi'^p$ and $\bigcirc^s_\chi \psi \notin \Phi'^c$, so $\Phi' \in F_{\bigcirc_\chi \psi}$.

Suppose $\bigcirc^s_\chi \psi \in \Phi^p$ (case b)). Then, the thesis trivially follows from Lemmas 5.2 and 6.1.

[$\Leftarrow$] Suppose case a) holds. If the computation is accepting, it means a state $\Phi_F \in F$ such that $\bigcirc^s_\chi \psi \notin \Phi^c_F$ and $\bigcirc^s_\chi \psi \notin \Phi^p_F$ is visited infinitely often. Moreover, by rule 1 of Section 5.2, if $\bigcirc_\chi \psi \in \Phi^c$ then $\bigcirc^s_\chi \psi$ is present in the atom of a subsequent state, which is then pushed onto the stack. It is easy to see from the proof of Lemma 5.2 that $\bigcirc^s_\chi \psi$ remains either in the atom of the current state or in the stack until the OPA reaches the right context of the forward-maximal chain starting in i. Thus, for all states in between, either $\bigcirc^s_\chi \psi$ is present in their atom or, due to Lemma 6.1, in their pending part. If Φ_F is reached, it means a configuration such as C_f occurs, which implies $(w, i) \models \bigcirc_\chi \psi$, by Lemma 5.2.

Case b) directly implies the right part of the double implication of Lemma 5.2, so the left part $(w, i) \models \bigcirc_\chi \psi$ follows. □

6.2. Summary until ($\mathcal{U}^{\boxdot}$) operator

For this operator we keep the rules for the finite case, except for the acceptance conditions, which are defined as follows. For any formula $\psi\,\mathcal{U}^{\boxdot}\,\theta$ and set $\boxdot \subseteq \{\lessdot, \doteq, \gtrdot\}$, we introduce the Büchi acceptance set $F_{\psi\mathcal{U}^{\boxdot}\theta} \in \mathbf{F}$, such that $\Phi \in F_{\psi\mathcal{U}^{\boxdot}\theta}$ iff $\bigcirc^s_\chi(\psi\,\mathcal{U}^{\boxdot}\,\theta) \notin \Phi^c$, $\bigcirc^s_\chi(\psi\,\mathcal{U}^{\boxdot}\,\theta) \notin \Phi^p$ and either

1. $\psi\,\mathcal{U}^{\boxdot}\,\theta \notin \Phi$ or
2. $\theta \in \Phi$.

Note that the acceptance conditions for formula $\bigcirc_\chi(\psi\,\mathcal{U}^{\boxdot}\,\theta)$ are involved, and they prevent words with open chains that delay forever the satisfaction of $\psi\,\mathcal{U}^{\boxdot}\,\theta$ from being mistakenly accepted.

Lemma 6.3. *Let AP be a finite set of atomic propositions, $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$ an OP alphabet, ψ and θ two OPTL formulae on it, $\boxdot \subseteq \{\lessdot, \doteq, \gtrdot\}$, and $\mathcal{A}_{\psi\mathcal{U}^{\boxdot}\theta}$ an ωOPBA satisfying rules 1-5 of Section 5.2, 1-3 of Section 6, and those of Section 6.2. For any ω-word w on $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$ and position i in w, we have*

$$(w, i) \models \psi\,\mathcal{U}^{\boxdot}\,\theta$$

if and only if $\mathcal{A}_{\psi\mathcal{U}^{\boxdot}\theta}$ performs at least one accepting computation such that $\psi\,\mathcal{U}^{\boxdot}\,\theta \in \Phi_i$, where Φ_i is the state of the automaton before reading position i.

Proof. The correctness of the finite-case constraints for a weak version of $\psi\,\mathcal{U}^{\boxdot}\,\theta$ follows from Lemma 5.4. We need to prove that of the Büchi acceptance conditions.

[⇒] If $\psi\,\mathcal{U}^{\boxdot}\,\theta$ does not hold infinitely often, then condition 1 trivially holds. Otherwise, for each position i in which $\psi\,\mathcal{U}^{\boxdot}\,\theta$ holds, there exists an OP summary path from i to a position j in which θ holds. The state Φ_j of the automaton right before reading j contains θ. If $\bigcirc^s_\chi(\psi\,\mathcal{U}^{\boxdot}\,\theta) \notin \Phi^c_j$ and $\bigcirc^s_\chi(\psi\,\mathcal{U}^{\boxdot}\,\theta) \notin \Phi^p_j$, condition 2 applies. Otherwise, $\bigcirc^s_\chi(\psi\,\mathcal{U}^{\boxdot}\,\theta)$ may be in Φ^c_j or Φ^p_j because either $\bigcirc_\chi(\psi\,\mathcal{U}^{\boxdot}\,\theta)$ is a requirement of ψ or θ, or it just happens to hold in this computation. In the latter case, by construction there exists also a computation in which $\bigcirc_\chi(\psi\,\mathcal{U}^{\boxdot}\,\theta)$ is not present, and $\bigcirc^s_\chi(\psi\,\mathcal{U}^{\boxdot}\,\theta) \notin \Phi^c_j$ and $\bigcirc^s_\chi(\psi\,\mathcal{U}^{\boxdot}\,\theta) \notin \Phi^p_j$. In the former, if $\psi\,\mathcal{U}^{\boxdot}\,\theta$ holds in i, then the instance of $\psi\,\mathcal{U}^{\boxdot}\,\theta$ required by $\bigcirc_\chi(\psi\,\mathcal{U}^{\boxdot}\,\theta)$ also holds, and the same reasoning can be applied to the position j'' in which the OP summary path satisfying it ends. Since ψ and θ are finite, we eventually reach a state satisfying condition 2.

[⇐] If a state satisfying condition 1 is repeated infinitely often, then the fact that all instances of $\psi\,\mathcal{U}^{\boxdot}\,\theta$ are satisfied follows from the correctness of its weak version (Lemma 5.4). Suppose a state Φ satisfying 2 is repeated infinitely often. If $\bigcirc^s_\chi(\psi\,\mathcal{U}^{\boxdot}\,\theta) \notin \Phi^c_j$ and $\bigcirc^s_\chi(\psi\,\mathcal{U}^{\boxdot}\,\theta) \notin \Phi^p_j$, then no instance of $\bigcirc_\chi(\psi\,\mathcal{U}^{\boxdot}\,\theta)$ is pending. Therefore, any path of a previous instance of $\psi\,\mathcal{U}^{\boxdot}\,\theta$ must pass through the position read by the automaton after being in state Φ. Since $\theta \in \Phi$, θ holds in this position, and $\psi\,\mathcal{U}^{\boxdot}\,\theta$ is satisfied. □

6.3. Yield-precedence hierarchical until ($\mathcal{U}^{\uparrow}$) operator

The acceptance conditions for this operator are very similar to those for the $\bigcirc_\chi$ operator, since they both rely on stack symbols to propagate their requirements. To adapt the construction of Section 5.5 to ω-words, we add $\psi\,\mathcal{U}^{\uparrow}_s\,\theta \in \mathrm{Cl}_{stack}(\varphi)$ if $\psi\,\mathcal{U}^{\uparrow}\,\theta \in \mathrm{Cl}(\varphi)$. We define $F_{\psi\mathcal{U}^{\uparrow}\theta} \in \mathbf{F}$, the Büchi acceptance set for $\psi\,\mathcal{U}^{\uparrow}\,\theta$, so that for any $\Phi \in \mathrm{Atoms}(\varphi) \times \mathrm{Cl}_{stack}(\varphi)$, $\Phi \in F_{\psi\mathcal{U}^{\uparrow}\theta}$ iff $\psi\,\mathcal{U}^{\uparrow}_s\,\theta \notin \Phi^c$ and $\psi\,\mathcal{U}^{\uparrow}_s\,\theta \notin \Phi^p$. Thus, the auxiliary operator $\psi\,\mathcal{U}^{\uparrow}_s\,\theta$ is used to keep track of an unsatisfied instance of $\psi\,\mathcal{U}^{\uparrow}\,\theta$, in the same way $\bigcirc^s_\chi\psi$ is used for $\bigcirc_\chi\psi$. We prove the correctness of this construction below.

Lemma 6.4. *Let AP be a finite set of atomic propositions, $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$ an OP alphabet, ψ and θ two OPTL formulae on it, and $\mathcal{A}_{\psi\mathcal{U}^{\uparrow}\theta}$ an ωOPBA satisfying rules 1-4 of Section 5.5, and 1-3 of Section 6. For any ω-word $w = \#xyz$ on $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$, let position $i = |x| + 1$ in w: we have*

$$(w, i) \models \psi\,\mathcal{U}^{\uparrow}\,\theta$$

if and only if $\mathcal{A}_{\psi\mathcal{U}^{\uparrow}\theta}$ performs at least one computation such that either:

a) the automaton goes through a configuration $\langle yz, \Phi, \rho\rangle$ with $\psi\,\mathcal{U}^{\uparrow}\,\theta \in \Phi^c$ and $\psi\,\mathcal{U}^{\uparrow}_s\,\theta \notin \Phi^p$, and the computation is accepting;
b) the automaton passes from configuration $\langle yz, \Phi_0, \alpha\gamma\rangle$ with $\psi\,\mathcal{U}^{\uparrow}\,\theta \in \Phi^c_0$ and $\psi\,\mathcal{U}^{\uparrow}_s\,\theta \in \Phi^p_0$ to $\langle z, \Phi_n, \alpha'\gamma\rangle$ such that $\psi\,\mathcal{U}^{\uparrow}_s\,\theta \notin \Phi^c_n$, $|\alpha| = 1$, $|\alpha'| \leq 1$, and for all $1 \leq k \leq n$, $\psi\,\mathcal{U}^{\uparrow}_s\,\theta \in \Phi^c_k$ or $\psi\,\mathcal{U}^{\uparrow}_s\,\theta \in \Phi^p_k$.

Proof. [⇒] If $(w, i) \models \psi\, \mathcal{U}^{\uparrow}\, \theta$, by Lemma 5.5, $\mathcal{A}_{\psi \mathcal{U}^{\uparrow} \theta}$ reaches configuration $C_0 = \langle yz, \Phi, \alpha\gamma \rangle$ with $\psi\, \mathcal{U}^{\uparrow}\, \theta \in \Phi^c$ to a configuration $\langle z, \Phi', \alpha'\gamma \rangle$ such that $|\alpha| = 1$, $|\alpha'| \leq 1$, and $\psi\, \mathcal{U}_s^{\uparrow}\, \theta, \psi\, \mathcal{U}_{end}^{\uparrow}\, \theta \notin \Phi'^c$.

If $\psi\, \mathcal{U}_s^{\uparrow}\, \theta \notin \Phi^p$ (case a)), from the proof of Lemma 5.5 we can conclude that all states pushed on the stack after C_0 are popped before C_f, as we noted in the proof of Lemma 6.2. Thus, $\psi\, \mathcal{U}_s^{\uparrow}\, \theta \notin \Phi'^c$ and $\psi\, \mathcal{U}_s^{\uparrow}\, \theta \notin \Phi'^p$, so $\Phi' \in F_{\psi \mathcal{U}^{\uparrow} \theta}$.

If $\psi\, \mathcal{U}_s^{\uparrow}\, \theta \in \Phi^p$ (case b)), the thesis follows from Lemmas 5.5 and Lemma 6.1.

[⇐] Suppose case a) holds. From the proof of Lemma 5.5, we can see that if $\psi\, \mathcal{U}^{\uparrow}\, \theta$ is present in the atom of a state, then $\psi\, \mathcal{U}_s^{\uparrow}\, \theta$ is inserted into the next state, and kept either in the atom of the current state, or in that of a state on the stack, until configuration C_f is reached. If the computation is accepting, then a state Φ_F such that $\psi\, \mathcal{U}_s^{\uparrow}\, \theta \notin \Phi_F^c$ and $\psi\, \mathcal{U}_s^{\uparrow}\, \theta \notin \Phi_F^p$ is visited. By Lemma 6.1, this may only happen if $\psi\, \mathcal{U}_s^{\uparrow}\, \theta$ is not present in the atom of the current state, nor in the stack. We can conclude that a configuration such as C_f is reached, and $(w, i) \models \psi\, \mathcal{U}^{\uparrow}\, \theta$.

Case b) implies the right part of the double implication of 5.5, so $(w, i) \models \psi\, \mathcal{U}^{\uparrow}\, \theta$ follows. □

6.4. Correctness

We now prove the correctness of the ωOPBA inductively on the syntactic structure of the formula.

Theorem 6.5. *Given a finite set of atomic propositions AP, an OP alphabet $(\mathscr{P}(AP), M_{\mathscr{P}(AP)})$, an ω-word w on it, and an OPTL formula φ, the automaton built according to the procedure in this section is such that we have*

$$(w, 1) \models \varphi$$

if and only if it performs at least one accepting computation on w. Moreover, the size of such an automaton is at most $2^{O(|\varphi|)}$.

Proof. The proof is carried out similarly to Theorem 5.9, by induction on the syntactic structure of φ. For the future operators, correctness is proved in Lemmas 6.2, 6.3, 6.4. For $\bigcirc_\chi$ and $\mathcal{U}^{\uparrow}$, we prove that they hold iff either a computation of the automaton is accepting, or if it proceeds to a configuration in which the stack has the same length as before they hold, i.e. no auxiliary symbols related to the last instance of them are contained in the stack, but those of other instances are. In the latter case, correctness of the whole model checking process is due to the correctness of the outermost instance of such operators. For $\mathcal{S}^{\downarrow}$, we rely on Lemma 5.6 and on the correctness of the Büchi acceptance conditions for $\bigcirc_\chi$. For the past operators, Lemmas 5.3, 5.7 and 5.8 hold.

Using $\text{Atoms}(\varphi) \times \text{Cl}_{stack}(\varphi)$ as the set of states only causes a polynomial increase of the size of the ωOPBA, which remains exponential in $|\varphi|$. □

From an argument similar to the one we made for the finite case follows

Corollary 6.6. *The satisfiability problem for OPTL on ω-words is EXPTIME-complete.*

7. 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 convey 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 power, in particular concerning First-Order completeness, as it has been done for various temporal logics, including NWTL [12].

A strictly related open issue is the relative expressive power of MSO logic vs. the FO one for OPLs. Such a relation has been thoroughly investigated in the case of regular languages in a rich literature, including [3]: it has been shown that regular languages expressible in FO logic coincide with the classes of *non-counting* or *aperiodic* languages, of the *star-free* ones, and many more. Thomas and others [39–41], however, have shown that such characterizations do not extend to the natural structural generalization of regular languages, i.e., tree languages. We conjecture, instead, that similar equivalences could hold for OPLs by referring to their strings rather than to their trees. Such a possible result, joined with a FO-complete temporal logic would allow to extend most classical results on logic characterization and model checking of regular languages to the much wider class of OPLs.

On the side of practical application, we are planning the development and implementation of suitable model checking algorithms based on the theoretical procedures presented in this paper. We also believe that most of the logics used to define properties of subclasses of context-free languages, 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.

Declaration of competing interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

Acknowledgements

We express our gratitude to the anonymous reviewers for their careful reading of the article, as their observations and advice allowed us to greatly improve its readability.

References

[1] M. Chiari, D. Mandrioli, M. Pradella, Temporal logic and model checking for operator precedence languages, in: GandALF 2018, in: EPTCS, vol. 277, Open Publishing Association, 2018, pp. 161–175.
[2] J.R. Büchi, Weak second-order arithmetic and finite automata, Math. Log. Q. 6 (1–6) (1960) 66–92, https://doi.org/10.1002/malq.19600060105.
[3] R. McNaughton, S. Papert, Counter-Free Automata, MIT Press, Cambridge, USA, 1971.
[4] M. Frick, M. Grohe, The complexity of first-order and monadic second-order logic revisited, Ann. Pure Appl. Log. 130 (1–3) (2004) 3–31, https://doi.org/10.1016/j.apal.2004.01.007.
[5] E.A. Emerson, Temporal and modal logic, in: Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B), Elsevier, 1990, pp. 995–1072.
[6] R. Alur, D.L. Dill, A theory of timed automata, Theor. Comput. Sci. 126 (2) (1994) 183–235, https://doi.org/10.1016/0304-3975(94)90010-8.
[7] R. McNaughton, Parenthesis grammars, J. ACM 14 (3) (1967) 490–500, https://doi.org/10.1145/321406.321411.
[8] J. Thatcher, Characterizing derivation trees of context-free grammars through a generalization of finite automata theory, J. Comput. Syst. Sci. 1 (1967) 317–322, https://doi.org/10.1016/S0022-0000(67)80022-9.
[9] B. von Braunmühl, R. Verbeek, Input-driven languages are recognized in log n space, in: Proc. of the Symp. on Fundamentals of Computation Theory, in: LNCS, vol. 158, Springer, 1983, pp. 40–51.
[10] R. Alur, P. Madhusudan, Adding nesting structure to words, J. ACM 56 (3) (2009), https://doi.org/10.1145/1516512.1516518.
[11] R. Alur, K. Etessami, P. Madhusudan, A temporal logic of nested calls and returns, in: TACAS 2004, in: LNCS, vol. 2988, Springer, 2004, pp. 467–481.
[12] R. Alur, M. Arenas, P. Barceló, K. Etessami, N. Immerman, L. Libkin, First-order and temporal logics for nested words, Log. Methods Comput. Sci. 4 (4) (2008), https://doi.org/10.2168/LMCS-4(4:11)2008.
[13] R. Alur, S. Chaudhuri, P. Madhusudan, Software model checking using languages of nested trees, ACM Trans. Program. Lang. Syst. 33 (5) (2011) 15, https://doi.org/10.1145/2039346.2039347.
[14] O. Burkart, B. Steffen, Model checking for context-free processes, in: CONCUR '92, in: LNCS, vol. 630, Springer, 1992, pp. 123–137.
[15] I. Walukiewicz, Pushdown processes: games and model-checking, Inf. Comput. 164 (2) (2001) 234–263, https://doi.org/10.1006/inco.2000.2894.
[16] A. Bouajjani, J. Esparza, O. Maler, Reachability analysis of pushdown automata: application to model-checking, in: CONCUR 1997, in: LNCS, vol. 1243, Springer, 1997, pp. 135–150.
[17] J. Esparza, D. Hansel, P. Rossmanith, S. Schwoon, Efficient algorithms for model checking pushdown systems, in: CAV 2000, in: LNCS, vol. 1855, Springer, 2000, pp. 232–247.
[18] O. Kupferman, N. Piterman, M.Y. Vardi, Model checking linear properties of prefix-recognizable systems, in: CAV 2002, in: LNCS, vol. 2404, Springer, 2002, pp. 371–385.
[19] J. Esparza, A. Kučera, S. Schwoon, Model checking LTL with regular valuations for pushdown systems, Inf. Comput. 186 (2) (2003) 355–376, https://doi.org/10.1016/S0890-5401(03)00139-1.
[20] R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. Reps, M. Yannakakis, Analysis of recursive state machines, ACM Trans. Program. Lang. Syst. 27 (4) (2005) 786–818, https://doi.org/10.1145/1075382.1075387.
[21] A. Bouajjani, R. Echahed, P. Habermehl, On the verification problem of nonregular properties for nonregular processes, in: LICS '95, 1995, pp. 123–133.
[22] O. Kupferman, N. Piterman, M.Y. Vardi, Pushdown specifications, in: LPAR 2002, in: LNCS, vol. 2514, Springer, 2002, pp. 262–277.
[23] R.W. Floyd, Syntactic analysis and operator precedence, J. ACM 10 (3) (1963) 316–333, https://doi.org/10.1145/321172.321179.
[24] S. Crespi Reghizzi, D. Mandrioli, D.F. Martin, Algebraic properties of operator precedence languages, Inf. Control 37 (2) (1978) 115–133, https://doi.org/10.1016/S0019-9958(78)90474-6.
[25] A. Barenghi, S. Crespi Reghizzi, D. Mandrioli, F. Panella, M. Pradella, Parallel parsing made practical, Sci. Comput. Program. 112 (2015) 195–226, https://doi.org/10.1016/j.scico.2015.09.002.
[26] S. Crespi Reghizzi, D. Mandrioli, Operator precedence and the visibly pushdown property, J. Comput. Syst. Sci. 78 (6) (2012) 1837–1867, https://doi.org/10.1016/j.jcss.2011.12.006.
[27] V. Lonati, D. Mandrioli, F. Panella, M. Pradella, Operator precedence languages: their automata-theoretic and logic characterization, SIAM J. Comput. 44 (4) (2015) 1026–1088, https://doi.org/10.1137/140978818.
[28] C. Lautemann, T. Schwentick, D. Thérien, Logics for context-free languages, in: CSL '94, 1994, pp. 205–216.
[29] D. Mandrioli, M. Pradella, Generalizing input-driven languages: theoretical and practical benefits, Comput. Sci. Rev. 27 (2018) 61–87, https://doi.org/10.1016/j.cosrev.2017.12.001.
[30] R. Alur, D. Fisman, Colored nested words, in: LATA 2016, Springer, 2016, pp. 143–155.
[31] R. Alur, Marrying words and trees, in: PODS '07, ACM, 2007, pp. 233–242.
[32] D. Grune, C.J. Jacobs, Parsing Techniques: A Practical Guide, Springer, New York, 2008.
[33] C.A.R. Hoare, An axiomatic basis for computer programming, Commun. ACM 12 (10) (1969) 576–580, https://doi.org/10.1145/363235.363259.
[34] D. Abrahams, Exception-safety in generic components, in: Generic Programming, Springer, 2000, pp. 69–79.
[35] T. Jensen, D. Le Metayer, T. Thorn, Verification of control flow based security properties, in: Proc. '99 IEEE Symp. on Security and Privacy, 1999, pp. 89–103.
[36] D.M. Gabbay, I. Hodkinson, M. Reynolds, Temporal Logic: Mathematical Foundations and Computational Aspects, Oxford University Press, New York, NY, USA, 1994.
[37] P. Wolper, M.Y. Vardi, A.P. Sistla, Reasoning about infinite computation paths, in: SFCS '83, 1983, pp. 185–194.
[38] S. Crespi Reghizzi, L. Breveglieri, A. Morzenti, Formal Languages and Compilation, second edition, Texts in Computer Science, Springer, 2013.
[39] U. Heuter, First-order properties of trees, star-free expressions, and aperiodicity, RAIRO Theor. Inform. Appl. 25 (2) (1991) 125–145, https://doi.org/10.1051/ita/1991250201251.

[40] W. Thomas, Logical aspects in the study of tree languages, in: Proc. Ninth Colloquium on Trees in Algebra and Programming, Cambridge University Press, 1984, pp. 31–49.
[41] A. Potthoff, W. Thomas, Regular tree languages without unary symbols are star-free, in: Fundamentals of Computation Theory, Springer, 1993, pp. 396–405.