Linear Temporal Logics for Structured Context-Free Languages

Michele Chiari[1], Davide Bergamaschi[1], Dino Mandrioli[1], and Matteo Pradella[1,2]

[1] DEIB, Politecnico di Milano, Italy name.surname@polimi.it
[2] IEIIT, Consiglio Nazionale delle Ricerche

Abstract. The need to extend traditional temporal logics to express and prove properties typical of stack-based formalisms led, among others, to CaRet and NWTL on Visibly Pushdown Languages (VPL). Such formalisms support, e.g., model checking of procedural programs and other context-free languages (CFL).
To further and significantly extend their expressive power, we recently introduced the logic OPTL, based on Operator Precedence Languages (OPL) which cover a much wider subclass of CFL. In this communication we survey the latest developments of our work. We introduced a novel temporal logic, POTL, that redefines OPTL to be First-Order complete. Furthermore, POTL's semantics is better connected to the typical tree-structure of CFL while retaining the ability to reason about linear time. Besides the theoretical advancements, we are also moving steps toward the implementation of POTL model checking.

Keywords: Linear Temporal Logic · Operator-Precedence Languages · Model Checking · Visibly Pushdown Languages

1 Introduction

The need for specifying requirements in model checking with formalisms more expressive than classical Linear Temporal Logic (LTL) has motivated much research towards the development of formalisms capable of expressing context-free properties [6,7,16,14,15,12,8]. Most notable are those based on Nested Words [5], or Visibly Pushdown Languages (VPL, [4]), a class of structured deterministic context-free languages (CFL) slightly more general than Parenthesis Languages [19]. Temporal logics based on them, such as CaRet [3], and the First-Order (FO) complete NWTL [1], introduced temporal modalities to explicitly reason about the nested structure of CFL, and found applications in the verification of procedural programs.

Such logics, however, suffer from the limited generality of VPL with respect to general CFL. This restricts the nesting relation on which they reason to be one-to-one, which is enough for modeling the matching between function calls and returns in procedural programs. However, it is not adequate to represent more complex constructs such as exceptions and continuations, which need to be

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

#[**call**[[**han**[**call**[**call**[**call**]]]**exc**]**call ret**]**ret**]#

Simple chains: $^{\mathbf{call}}[\mathbf{call}]^{\mathbf{exc}}$, $^{\mathbf{call}}[\mathbf{han\,exc}]^{\mathbf{call}}$
Composed chains: $^{\mathbf{call}}[\mathbf{call}[\mathbf{call}]]^{\mathbf{exc}}$
$^{\mathbf{call}}[[\mathbf{han}[\mathbf{call}[\mathbf{call}[\mathbf{call}]]]\mathbf{exc}]\mathbf{call\ ret}]^{\mathbf{call}}$

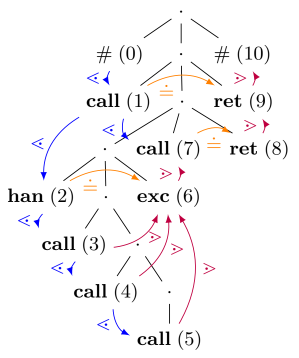


Fig. 1. OPM $M_{\mathbf{call}}$ (top left); a word and its subdivision in chains according to $M_{\mathbf{call}}$, with some examples of simple and composed chains in it (bottom left); the ST corresponding to the word (right). In the ST, dots represent non-terminals, and PR are show by colored arrows.

modeled by one-to-many or many-to-one relations. E.g., an exception is a single event that needs to be put in relation with all function instances it terminates.

To further expand the expressiveness of temporal logics we introduced OPTL [9], a temporal logic based on Operator Precedence Languages (OPL, [13]), a class of structured CFL which retains all closure and decidability properties needed for model checking. OPL are wider than VPL [11], and their more general nesting relation, called the *chain* relation, can be many-to-one or one-to-many.

One of the features that is generally expected from temporal logics is equivalence to FO Logic. Since proving this property for OPTL seems arduous, as is for CaRet, in [10] we surveyed some possible ways to define a logic more strictly related to the context-free structure of OPL, for which FO-completeness could be proved. Thus, we devised Precedence Oriented Temporal Logic (POTL), which makes reasoning on the underlying syntax tree of an OPL word easier, while remaining a linear-time temporal logic. We gave a FO-completeness proof of POTL on finite words, which can be extended to ω-words by composition arguments. Moreover, the automata-theoretic model checking procedure we devised for POTL has the same asymptotic complexity of less expressive formalisms, being exponential in formula length.

2 Operator Precedence Languages

OPL have been inspired by precedence relations among operators in arithmetic expressions parsing. They are generated by grammars in operator form, i.e. whose rules' right-hand sides (rhs) have no consecutive non-terminals. Their parsers are guided in recognizing and reducing grammar rhs by three binary precedence relations (PR) among terminal symbols. Given two terminals a, b, for any non-terminals A, B, C and mixed terminal/non-terminal strings α, β, γ, we say *a yields precedence* to b ($a \lessdot b$) if there exists a rule $A \to \alpha a C \beta$, s.t. a string $B b\gamma$ or $b\gamma$ derives from C in any number of passes; a is *equal in precedence* to b ($a \doteq b$) if there exists a rule $A \to \alpha a C b \beta$ or $A \to \alpha a b \beta$; and *a takes precedence over b* ($a \gtrdot b$) if there is a rule $A \to \alpha C b \beta$, s.t. $\gamma a B$ or γa derives from C. In practice, $a \lessdot b$ if b is the beginning of a rhs; $a \doteq b$ if they belong to the same rhs; $a \gtrdot b$ if a is the end of a rhs. If at most one PR holds between any terminal pair, once all PR are collected into an operator precedence matrix (OPM), the syntax tree (ST) of any word on the same alphabet is fully determined.

The way PR determine the ST of a string is formalized by *chains*:

Definition 1. *A* simple chain ${}^{c_0}[c_1 c_2 \dots c_\ell]^{c_{\ell+1}}$ *is a string* $c_0 c_1 c_2 \dots c_\ell c_{\ell+1}$*, such that:* $c_0, c_{\ell+1} \in \Sigma \cup \{\#\}$*,* $c_i \in \Sigma$ *for every* $i = 1, 2, \dots \ell$ *(*$\ell \geq 1$*), 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* $c_0 s_0 c_1 s_1 c_2 \dots c_\ell s_\ell c_{\ell+1}$*, where* ${}^{c_0}[c_1 c_2 \dots c_\ell]^{c_{\ell+1}}$ *is a simple chain, and* $s_i \in \Sigma^*$ *is the empty string or is such that* ${}^{c_i}[s_i]^{c_{i+1}}$ *is a chain (simple or composed), for every* $i = 0, 1, \dots, \ell$ *(*$\ell \geq 1$*). Such a composed chain will be written as* ${}^{c_0}[s_0 c_1 s_1 c_2 \dots c_\ell s_\ell]^{c_{\ell+1}}$*.* c_0 *(resp.* $c_{\ell+1}$*) is called its* left *(resp.* right*)* context*.*

Fig. 1 shows OPM $M_{\mathbf{call}}$, together with word # **call han call call call exc call ret ret** #. Its chain structure is shown by surrounding chain bodies with brackets. The word is delimited by #, s.t. $\# \lessdot a$ and $a \gtrdot \#$ for any terminal a. In the ST, each simple chain body corresponds to a rhs in the tree, and composed chains contain non-terminals, which are the bodies of other simple or composed chains. Thus, the structure of chains in a given string is isomorphic to its ST.

OPL also have a defining class of pushdown automata, Operator Precedence Automata [17]. We use them for model checking POTL, but we omit their definition for lack of space. For a better definition of OPL we refer readers to [18].

3 Precedence Oriented Temporal Logic

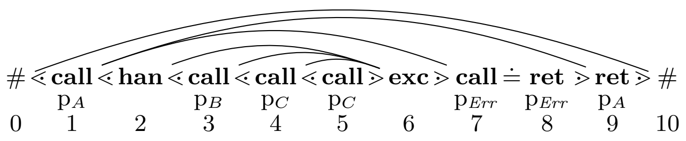
Given a finite set of atomic propositions AP, the syntax of POTL follows:

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

where $\mathrm{a} \in AP$, and $t \in \{d, u\}$.

The semantics of POTL is based on the *word structure* –also called *OP word* for short– $\langle U, M_{\mathcal{P}(AP)}, P \rangle$, where $U = \{0, 1, \dots, n, n+1\}$, with $n \in \mathbb{N}$ is a set of word positions; $M_{\mathcal{P}(AP)}$ is an OPM on $\mathcal{P}(AP)$; $P \colon U \to \mathcal{P}(AP)$ is a function associating each word position in U with the set of atomic propositions that hold in that position, with $P(0) = P(n+1) = \{\#\}$.

We use a partitioning of AP into a set of normal propositional labels (in round font), and *structural labels* (SL, in bold). SL define the OP structure of the word: $M_{\mathcal{P}(AP)}$ is only defined for subsets of AP containing exactly one SL, so that given two SL $\mathbf{l}_1, \mathbf{l}_2$, for any $a, a', b, b' \in \mathcal{P}(AP)$ s.t. $\mathbf{l}_1 \in a, a'$ and $\mathbf{l}_2 \in b, b'$ we have $M_{\mathcal{P}(AP)}(a, b) = M_{\mathcal{P}(AP)}(a', b')$. This way, we define an OPM on $\mathcal{P}(AP)$ by only giving relations between SL, as we did for $M_{\mathbf{call}}$. Given two positions i, j and a PR $\pi \in \{\lessdot, \doteq, \gtrdot\}$, we write $i \;\pi\; j$ to say $P(i) \;\pi\; P(j)$.



$\# \lessdot \mathbf{call} \lessdot \mathbf{han} \lessdot \mathbf{call} \lessdot \mathbf{call} \lessdot \mathbf{call} \gtrdot \mathbf{exc} \gtrdot \mathbf{call} \doteq \mathbf{ret} \gtrdot \mathbf{ret} \gtrdot \#$
p_A p_B p_C p_C p_{Err} p_{Err} p_A
0 1 2 3 4 5 6 7 8 9 10

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

We define the chain relation $\chi \subseteq U \times U$ so that $\chi(i, j)$ holds between two positions i, j iff $i < j - 1$, and i and j are resp. the left and right contexts of the same chain. For composed chains, χ may not be one-to-one, but also one-to-many or many-to-one. Fig. 2 shows the execution trace of a procedural program. The χ relation is meaningful w.r.t. the program semantics: every **call** to a function is in relation with the **ret** terminating it, and all instructions issued by that function are contained between them. **call**s terminated by an exception are in relation with the corresponding **exc** statement, so the chain relation is many-to-one.

The truth of POTL formulas is defined w.r.t. a single word position. 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. Next, while giving the formal semantics of POTL operators, we illustrate it by showing how it can be used to express properties on program execution traces, such as the one of Fig. 2.

Next/back operators. The *downward* next and back operators $\bigcirc^d$ and $\circleddash^d$ are like their LTL counterparts, except they are true only if the next (resp. current) position is at a lower or equal ST level than the current (resp. preceding) one. The *upward* next and back, $\bigcirc^u$ and $\circleddash^u$, are symmetric. Formally, $(w, i) \models \bigcirc^d \varphi$ iff $(w, i+1) \models \varphi$ and $i \lessdot (i+1)$ or $i \doteq (i+1)$, and $(w, i) \models \circleddash^d \varphi$ iff $(w, i-1) \models \varphi$, and $(i-1) \lessdot i$ or $(i-1) \doteq i$. Substitute $\lessdot$ with $\gtrdot$ to obtain the semantics for $\bigcirc^u$ and $\circleddash^u$. E.g., $\bigcirc^d$**call** means that the next position is an inner call (it holds in pos. 2, 3, 4 of Fig. 2), $\circleddash^d$**call** to say that the previous position is a **call**, and the current is the first of the body of a function (pos. 2, 4, 5), or the **ret** of an empty one (pos. 8).

The *chain* next and back operators χ_F^t and χ_P^t evaluate their argument respectively on future and past positions in the chain relation with the current one. The *downward* (resp. *upward*) variant only considers chains whose right context goes down (resp. up) in the ST. $(w, i) \models \chi_F^d \varphi$ iff there exists a position $j > i$ such that $\chi(i, j)$, $i \lessdot j$ or $i \doteq j$, and $(w, j) \models \varphi$. $(w, i) \models \chi_P^d \varphi$ iff there exists a position $j < i$ such that $\chi(j, i)$, $j \lessdot i$ or $j \doteq i$, and $(w, j) \models \varphi$. Replace $\lessdot$ with $\gtrdot$ for the upward versions. E.g., in pos. 1 of Fig. 2, $\chi_F^d \mathrm{p}_{Err}$ holds because $\chi(1, 7)$, meaning that p_A calls p_{Err} at least once. Also, χ_F^u**exc** is true in **call** positions whose procedure is terminated by an exception thrown by an inner procedure (e.g. pos. 3 and 4). χ_P^u**call** is true in **exc** statements that terminate at least one procedure other than the one raising it, such as the one in pos. 6. χ_F^d**ret** and χ_F^u**ret** hold in **call**s to non-empty procedures that terminate normally, and not due to an uncaught exception (e.g., pos. 1).

Until/Since operators. The *summary* until $\psi \mathcal{U}_\chi^t \theta$ (resp. since $\psi \mathcal{S}_\chi^t \theta$) operator is obtained by inductively applying the $\bigcirc^t$ and χ_F^t (resp. $\ominus^t$ and χ_P^t) operators. It holds in a position if either θ holds, or ψ holds with $\bigcirc^t(\psi \mathcal{U}_\chi^t \theta)$ (resp. $\ominus^t(\psi \mathcal{S}_\chi^t \theta)$) or $\chi_F^t(\psi\ \mathcal{U}_\chi^t\ \theta)$ (resp. $\chi_P^t(\psi\ \mathcal{S}_\chi^t\ \theta)$). It is an until on paths that move not only between consecutive positions, but also between contexts of a chain, skipping its body. With OPM $M_{\mathbf{call}}$, this means skipping function bodies. The downward variants move between positions at the same level in the ST (i.e., in the same simple chain body), or down in the nested chain structure. The upward ones move at the same or to higher levels of the ST. Formula $\top\, \mathcal{U}_\chi^u$ **exc** is true in positions contained in the frame of a function that is terminated by an exception. It is true in pos. 3 of Fig. 2 because of path 3-6, and false in pos. 1, because no path can enter chain $\chi(1, 9)$. Formula $\top \mathcal{U}_\chi^d$**exc** is true in call positions whose function frame contains **exc**s, but that are not directly terminated by one of them, such as the one in pos. 1 (with path 1-2-6). Moreover, **call** $\mathcal{U}_\chi^d$ (**ret** $\wedge\, \mathrm{p}_{Err}$) holds in pos. 1 because of path 1-7-8, (**call** $\vee$ **exc**) $\mathcal{S}_\chi^u\, \mathrm{p}_B$ in pos. 7 because of path 3-6-7, and (**call** $\vee$ **exc**) $\mathcal{U}_\chi^u$ **ret** in 3 because of path 3-6-7-8.

Hierarchical Operators These operators enable reasoning on multiple positions in the chain relation with a single one. The upward and downward hierarchical next are defined as $(w, i) \models \bigcirc_H^u \varphi$ iff there exist a position $h < i$ s.t. $\chi(h, i)$ and $h \lessdot i$ and a position $j = \min\{k \mid i < k \wedge \chi(h, k) \wedge h \lessdot k\}$ and $(w, j) \models \varphi$; $(w, i) \models \bigcirc_H^d \varphi$ iff there exist a position $h > i$ s.t. $\chi(i, h)$ and $i \gtrdot h$ and a position $j = \min\{k \mid i < k \wedge \chi(k, h) \wedge k \gtrdot h\}$ and $(w, j) \models \varphi$. Their past counterparts are symmetric, and their until and since operators are obtained by iterating them.

We proved the following claims on POTL's expressivity:

Theorem 1. *POTL = FO with one free variable on finite OP words.*

Corollary 1. *NWTL $\subset$ OPTL $\subseteq$ POTL over finite OP words.*

Moreover, we developed an automata-theoretic model checking procedure, whose complexity is not asymptotically greater than comparable formalisms:

Theorem 2. *Given a POTL formula φ, it is possible to build an OPA $\mathcal{A}_\varphi$ accepting the language denoted by φ with at most $2^{O(|\varphi|)}$ states.*

$\mathcal{A}_\varphi$ can then be intersected [17] with an OPA modeling a program, and emptiness can be decided with *summarization* techniques [2].

4 Conclusions

We surveyed our work on POTL, a novel temporal logic based on OPL, for which we proved FO-completeness. We gave a model-checking procedure based on automata construction, which we implemented in a prototype tool. We plan to further develop such tool to apply POTL to verification tasks.

References

1. Alur, R., Arenas, M., Barceló, P., Etessami, K., Immerman, N., Libkin, L.: First-order and temporal logics for nested words. LMCS **4**(4) (2008). https://doi.org/10.2168/LMCS-4(4:11)2008
2. Alur, R., Bouajjani, A., Esparza, J.: Model checking procedural programs. In: Handbook of Model Checking, pp. 541–572. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_17
3. Alur, R., Etessami, K., Madhusudan, P.: A temporal logic of nested calls and returns. In: TACAS 2004. pp. 467–481. Springer (2004). https://doi.org/10.1007/978-3-540-24730-2_35
4. Alur, R., Madhusudan, P.: Visibly Pushdown Languages. In: ACM STOC (2004)
5. Alur, R., Madhusudan, P.: Adding nesting structure to words. JACM **56**(3) (2009). https://doi.org/10.1145/1516512.1516518
6. Bouajjani, A., Echahed, R., Habermehl, P.: On the verification problem of non-regular properties for nonregular processes. In: LICS 95. pp. 123–133 (1995). https://doi.org/10.1109/LICS.1995.523250
7. Bouajjani, A., Habermehl, P.: Constrained properties, semilinear systems, and petri nets. In: CONCUR '96, Concurrency Theory, 7th International Conference, Pisa, Italy, August 26-29, 1996, Proceedings. LNCS, vol. 1119, pp. 481–497. Springer (1996). https://doi.org/10.1007/3-540-61604-7_71
8. Chatterjee, K., Ma, D., Majumdar, R., Zhao, T., Henzinger, T.A., Palsberg, J.: Stack size analysis for interrupt-driven programs. Inf. Comput. **194**(2), 144–174 (2004). https://doi.org/10.1016/j.ic.2004.06.001
9. Chiari, M., Mandrioli, D., Pradella, M.: Temporal logic and model checking for operator precedence languages. In: GandALF 2018. EPTCS, vol. 277, pp. 161–175. Open Publishing Association (2018). https://doi.org/10.4204/EPTCS.277.12
10. Chiari, M., Mandrioli, D., Pradella, M.: Word- and tree-based temporal logics for operator precedence languages. In: Proc. 20th Italian Conference on Theoretical Computer Science, ICTCS 2019, Como, Italy, September 9-11, 2019. CEUR Workshop Proceedings, vol. 2504, pp. 222–228. CEUR-WS.org (2019), http://ceur-ws.org/Vol-2504/paper25.pdf
11. Crespi Reghizzi, S., Mandrioli, D.: Operator Precedence and the Visibly Pushdown Property. JCSS **78**(6), 1837–1867 (2012). https://doi.org/10.1016/j.jcss.2011.12.006
12. Esparza, J., Kučera, A., Schwoon, S.: Model checking LTL with regular valuations for pushdown systems. Information and Computation **186**(2), 355–376 (2003). https://doi.org/10.1016/S0890-5401(03)00139-1
13. Floyd, R.W.: Syntactic Analysis and Operator Precedence. JACM **10**(3), 316–333 (1963). https://doi.org/10.1145/321172.321179
14. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic, pp. 99–217. Springer (2002). https://doi.org/10.1007/978-94-017-0456-4_2
15. Jensen, T., Le Metayer, D., Thorn, T.: Verification of control flow based security properties. In: Proc. '99 IEEE Symp. on Security and Privacy. pp. 89–103 (1999). https://doi.org/10.1109/SECPRI.1999.766902
16. Kupferman, O., Piterman, N., Vardi, M.Y.: Pushdown Specifications. In: LPAR 2002. LNCS, vol. 2514, pp. 262–277. Springer (2002). https://doi.org/10.1007/3-540-36078-6_18
17. Lonati, V., Mandrioli, D., Panella, F., Pradella, M.: Operator precedence languages: Their automata-theoretic and logic characterization. SIAM J. Comput. **44**(4), 1026–1088 (2015). https://doi.org/10.1137/140978818

18. Mandrioli, D., Pradella, M.: Generalizing input-driven languages: Theoretical and practical benefits. Computer Science Review **27**, 61–87 (2018). https://doi.org/10.1016/j.cosrev.2017.12.001
19. McNaughton, R.: Parenthesis Grammars. JACM **14**(3), 490–500 (1967). https://doi.org/10.1145/321406.321411