Word- and Tree-Based Temporal Logics for Operator Precedence Languages***

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

[1] DEIB, Politecnico di Milano – Piazza L. Da Vinci, 32, 20133, Milano, Italy
{name.surname}@polimi.it
[2] IEIIT, Consiglio Nazionale delle Ricerche

Abstract. In recent years we resumed the study of operator precedence languages (OPL), inspired by their algebraic and parallel parsability properties. OPL significantly generalize visibly pushdown languages in expressive power. We have shown that OPL enjoy a characterization in terms of monadic second order logic in the style of the traditional one for regular languages. We are investigating a characterization of a subclass of OPL in terms of first-order logic, retracing the literature on regular languages. The algebraic properties of OPL naturally suggested a further step, devoted to devising model checking techniques for them. In this communication we briefly report on early steps already moved in this direction and we outline a roadmap for next developments. In doing so, we plan to exploit a distinguishing feature of OPL which allows to manage them both as string languages and as tree-structured languages, thus joining two traditional but different approaches to the study of context-free languages. We expect to achieve a temporal logic-style language that is well suited to express properties of systems in application fields wider than those based on regular languages. We anticipate to obtain model-checking procedures with comparable performances.

Keywords: Temporal Logic · Operator Precedence Languages · Input-Driven Languages · Visibly Pushdown Languages · Model Checking.

1 Introduction

Formal correctness verification consists in proving, preferably automatically, that a computer system –or precisely, a model thereof– satisfies its requirement specification. The variety of properties constituting the specification depends on the mathematical formalisms employed to model the system, and to write the specification. Formalisms based on the limited, yet practically important, finite state machines (FSM) have lead to completely automated verification processes. This could not be achieved by the Turing-complete ones, due to their undecidability.

Monadic second-order logic (MSO) [11] was introduced to give a logical characterization to regular languages, also suitable for requirement specification. Unfortunately, the verification problem for both MSO and its first-order fragment

* Work partially supported by project AUTOVAM (funded by Fondazione Cariplo and Regione Lombardia).

(MFO) is computationally intractable [19,26]. Thus, A. Pnueli, E. Clarke, E.A. Emerson and J.H. Alpern introduced temporal logics: Linear Temporal Logic (LTL) [27], and its branching-time counterparts, such as CTL* [15]. Model checking temporal logic formulas is PSPACE complete [28], with a time complexity bounded by a single exponential in their length, definitely more acceptable.

LTL was proved to coincide in expressive power with the first-order fragment of regular languages. Despite being a significant assurance of expressive power, this left room for improvement. This purpose lead to the birth of a number of variants of temporal logics, such as timed [3] versions. Another natural path for improvement leads upwards in the Chomsky hierarchy. Unfortunately, context-free languages (CFL) lack closure and decidability properties necessary for model checking. The first attempts consisted in model checking regular properties of pushdown systems [10,16,20,17,6], without tackling, however, the problem of verification against truly context-free based specifications. After a few early works in this direction [9,21], a significant further step has been the introduction of the temporal logic CaRet [7], by R. Alur and P. Madhusudan. CaRet is based on Visibly Pushdown Languages (VPL) [4], a subclass of Deterministic CFL (DCFL) that enjoys all the properties required for model checking (they form a Boolean algebra), but sacrificing expressive power. VPL were also characterized in terms of MSO [5], and their first-order fragment was isolated by introducing the temporal logic NWTL, and its model checking procedure [2].

We hereby report on our endeavors to obtain greater expressiveness by model checking an even wider language class: Operator Precedence Languages (OPL) [18]. OPL were originally introduced by Floyd in the context of programming language parsing.[3] Thus, despite not capturing the full class of DCFL, they can express the word structures most typical of CFL. Their being inspired by precedence relations between operators in arithmetic expressions allows them to represent tree-like structures not immediately visible in words, so as string recognition is not necessarily real-time. This greatly differentiates OPL form VPL, whose nesting structure is immediately *visible* and determined by a partition on the terminal alphabet, making them only a little more general than parenthesis languages [25]. The structure of OPL, instead, is determined by the precedence relations between pairs of terminals: OPL embrace a greater expressive power, while not sacrificing their closure properties [14,13].

Initially, we developed their MSO characterization [23], by extending the matching relation introduced in [22]. Then, we introduced the temporal logic OPTL [12], an attempt at generalizing the approach introduced with NWTL to the more complex algebraic structures allowed by OPL. Our findings in this respect are summarized in Section 3, after an overview of OPL in Section 2. The structure of OPL words is, however, general to a point that their tree-like nature is predominant. We thus also consider approaches that are more common in tree logics, such as Conditional XPath. We report on such aims in Section 4. In Section 5 we comment on future research steps.

[3] We devised efficient parallel parsers for OPL [8], but this is out of the scope of this paper.

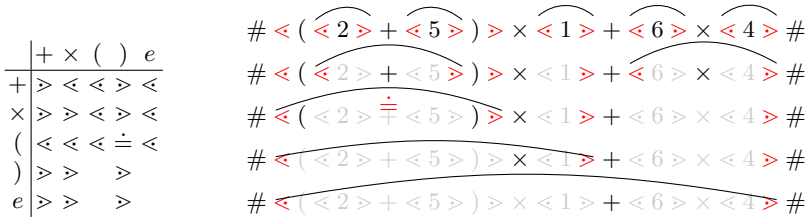

Fig. 1: Parsing of arithmetic expression (right, OPM on the left). e represents any number. The *chain* relation is shown with lines connecting rhs contexts.

2 Operator Precedence Languages

OPL are inspired by precedence relations among operators in arithmetic expressions. They are generated by grammars in operator form, i.e. whose rules' right-hand sides (rhs) have no consecutive non-terminals. Three binary precedence relations (PR) among terminal symbols derive from them, guiding their parsers in recognizing and reducing grammar rhs. 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 $Ba\gamma$ or $a\gamma$ 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 abstract syntax tree (AST) of any word on the same alphabet is fully determined. Fig. 1 shows the parsing steps of an OPL of arithmetic expressions, with PR between consecutive terminals. The word is delimited by #, s.t. $\# \lessdot a$ and $a \gtrdot \#$ for any terminal a. We call *context* the two terminals surrounding a rhs. In the OPM, we have $+ \lessdot \times$, and $\times \gtrdot +$. So, rhs with $\times$ are reduced before $+$, reflecting the precedence of $\times$ w.r.t. $+$ in the AST. Thus, the OPM induces a *matching relation* between the contexts of the same rhs, also called the *chain* relation. It completely defines the tree structure of a string [23], playing the same role as the matching relation μ in nested words [5]. Being based on VPL, μ can only be one-to-one, and a position cannot be both the right and left side of the relation. Both constraints are lifted for the chain relation, making it much more general.

Nested words have been introduced with the main purpose of modeling procedural programs. OPL can model them too, with the addition of features that cannot be captured by VPL, such as exceptions. An exception is an event that forces the termination of multiple functions, until a handler blocks the stack unwinding. This process can be modeled as in Fig. 2. OPM M_{call} determines the chain relation, shown by arcs connecting related positions. Every *call* to a function is in relation with the *return* terminating it, and all instructions issued

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

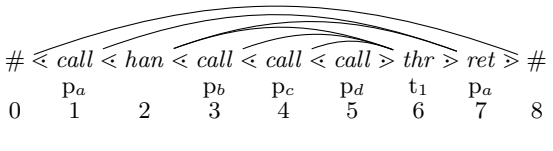

Fig. 2: Example of execution trace (right), according to OPM M_{call} (left). Procedure p_a is called in pos. 1, and it installs an exception handler (*han*) in 2. Then, three nested procedures are called, the innermost one (p_d) throwing an exception (*thr*), caught by the handler. Finally, p_a returns, uninstalling the handler.

by that function are contained between them. Some invocations are, however, terminated by an exception. Their *calls* are in relation with the corresponding *throw* statement. The chain relation is thus many-to-one in this case.

3 Word-Based Linear Temporal Logic

This approach is inspired to logics such as NWTL, following the linear structure of words more than their tree structure. We first explored it with the logic OPTL in [12], where an automata-theoretic model checking procedure is also sketched.

A natural class of properties that can be expressed in OPTL are Hoare-style pre/post-conditions. This can be achieved by means of an operator that asserts a formula in a position in the chain relation with the current one. This operator can refer to a single position or to a collection thereof by means of suitable quantifications. In OPTL, this role is taken by the *matching next* $\bigcirc_\chi$ operator, which considers the maximal (outermost) chain and selects the farthest position in the chain relation with the current one. In Fig. 2, the maximal chain starting from pos. 2 is 7, and not 6. For instance, the OPTL formula $\varphi \equiv \Box[\rho \wedge \mathit{call} \implies \bigcirc_\chi \theta]$, where $\Box$ is the LTL *globally* operator, says that whenever precondition ρ holds at the point of a procedure call, post-condition θ will hold at the corresponding *return* or *throw*, extending this kind of properties to exceptions. Referring to the C++ programming language, if θ is a class invariant asserting that an object is in a valid state, φ expresses *weak exception safety* (i.e. the exception leaves the object in a valid state) [1]. If $\rho \equiv \theta$ represents the whole state of the object, then φ expresses strong exception safety (i.e. the exception leaves the object untouched). Alternatively, the matching next could be replaced by an operator that quantifies existentially on positions in the chain relation with the current one, without choosing one of them a priori.

Both OPTL's and NWTL's *until* and *since* operators add to the traditional LTL ones, named *linear*, the so-called *summary* ones which skip all positions between a call and its matching return. In OPL words, however, jumping from a position to a matching one can lead to different points since multiple positions can be in the chain relation with a single one, so only one of them must be chosen.

Thus, consistently with the previous definition of the matching next operator, the farthest (maximal) one is chosen, but other criteria may be used.

In OPL words, PR also need to be taken into account. Paths can be better controlled by admitting only some of the PR between subsequent positions in the path. Admitting the $\lessdot$ relation lets paths enter chain bodies, going downwards in the AST; symmetrically, $\gtrdot$ exit them, going upwards in the AST. In Fig. 2, until operators based on summary paths not allowing the $\gtrdot$ relation are limited to the function body in which the operator is evaluated. This allows to express function-local properties, such as "function A will never write to variable x", including or excluding subcalls.

4 Tree-Based Linear Temporal Logic

Another possible approach is to adapt the first-order complete tree-logic Conditional XPath [24] to OPL words, by applying it to their AST, thus seeing OP words from the point of view of their tree-like structure, rather than from that of linear words. Since the tree structure of OPL words is given by the chain relation, this logic evaluates its formulas on *pairs of positions* that are either consecutive or in the chain relation, called *contexts*, instead of single positions. A pair of until/since operators allows for "vertical" movement from a context to those immediately contained into it. This means movement in the AST from a non-terminal to one of its non-terminal children. Another pair of until/since operators enables movement among consecutive contexts in the same rhs, i.e. among the non-terminals in the same rhs.

Applying such a logic to procedural program specification (Fig. 2), it is easier to express properties that vertically analyze stack traces, such as stack inspection, extending the capabilities of VPL-based logics to programs with exceptions. For example, properties such as "function C should be called only when B is on stack, with no intervening call to A", or "if a procedure D is terminated by an uncaught exception, B must be terminated by the same exception as well, and A cannot occur between them" are easily expressible with a vertical since operator. In Fig. 2, they are witnessed resp. by the context-paths $(4,6),(3,6)$ and $(5,6),(4,6),(3,6)$. A vertical until may express properties on sub-calls in a function frame (e.g. "function A never calls B"), and horizontal operators on instructions in the current frame only (e.g. "proc. A never writes to variable x").

5 Conclusion

In this short paper, we reviewed our previous work on the matter of developing temporal logic and model checking on OPL, and describe the premises for further developments. Our future efforts will be aimed at better formally defining the two possible approaches we hinted, by investigating their expressiveness, the relationships between them, and by conceiving, implementing, and applying practical model checking procedures.

References

1. Abrahams, D.: Exception-Safety in Generic Components. In: Generic Programming. pp. 69–79. Springer Berlin Heidelberg (2000). https://doi.org/10.1007/3-540-39953-4_6
2. Alur, R., Arenas, M., Barceló, P., Etessami, K., Immerman, N., Libkin, L.: First-order and temporal logics for nested words. Logical Methods in Computer Science **4**(4) (2008). https://doi.org/10.2168/LMCS-4(4:11)2008
3. Alur, R., Dill, D.L.: A Theory of Timed Automata. Theor. Comput. Sci. **126**(2), 183–235 (1994). https://doi.org/10.1016/0304-3975(94)90010-8
4. Alur, R., Madhusudan, P.: Visibly Pushdown Languages. In: STOC: ACM Symposium on Theory of Computing (STOC) (2004)
5. Alur, R., Madhusudan, P.: Adding nesting structure to words. JACM **56**(3) (2009). https://doi.org/10.1145/1516512.1516518
6. Alur, R., Benedikt, M., Etessami, K., Godefroid, P., Reps, T., Yannakakis, M.: Analysis of recursive state machines. ACM Trans. Program. Lang. Syst. **27**(4), 786–818 (Jul 2005). https://doi.org/10.1145/1075382.1075387
7. Alur, R., Etessami, K., Madhusudan, P.: A temporal logic of nested calls and returns. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004). LNCS, vol. 2988, pp. 467–481. Springer Berlin Heidelberg (2004)
8. Barenghi, A., Crespi Reghizzi, S., Mandrioli, D., Panella, F., Pradella, M.: Parallel parsing made practical. Sci. Comput. Program. **112**, 195–226 (2015). https://doi.org/10.1016/j.scico.2015.09.002
9. Bouajjani, A., Echahed, R., Habermehl, P.: On the verification problem of nonregular properties for nonregular processes. In: Proceedings of Tenth Annual IEEE Symposium on Logic in Computer Science. pp. 123–133 (June 1995). https://doi.org/10.1109/LICS.1995.523250
10. Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: Application to model-checking. In: CONCUR 1997. LNCS, vol. 1243, pp. 135–150. Springer Berlin Heidelberg (1997). https://doi.org/10.1007/3-540-63141-0_10
11. Büchi, J.R.: Weak Second-Order Arithmetic and Finite Automata. Mathematical Logic Quarterly **6**(1-6), 66–92 (1960). https://doi.org/10.1002/malq.19600060105
12. Chiari, M., Mandrioli, D., Pradella, M.: Temporal logic and model checking for operator precedence languages. In: Proceedings Ninth International Symposium on Games, Automata, Logics, and Formal Verification, Saarbrücken, Germany, 26-28th September 2018. Electronic Proceedings in Theoretical Computer Science, vol. 277, pp. 161–175. Open Publishing Association (2018). https://doi.org/10.4204/EPTCS.277.12
13. 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
14. Crespi Reghizzi, S., Mandrioli, D., Martin, D.F.: Algebraic Properties of Operator Precedence Languages. Information and Control **37**(2), 115–133 (May 1978). https://doi.org/10.1016/S0019-9958(78)90474-6
15. Emerson, E.A., Halpern, J.Y.: "Sometimes" and "Not Never" Revisited: on Branching Versus Linear Time Temporal Logic. J. ACM **33**(1), 151–178 (Jan 1986). https://doi.org/10.1145/4904.4999
16. Esparza, J., Hansel, D., Rossmanith, P., Schwoon, S.: Efficient algorithms for model checking pushdown systems. In: Computer Aided Verification (CAV 2000). LNCS, vol. 1855, pp. 232–247. Springer Berlin Heidelberg (2000)

17. Esparza, J., Kučera, A., Schwoon, S.: Model checking LTL with regular valuations for pushdown systems. Information and Computation **186**(2), 355–376 (nov 2003). https://doi.org/10.1016/S0890-5401(03)00139-1
18. Floyd, R.W.: Syntactic Analysis and Operator Precedence. JACM **10**(3), 316–333 (1963). https://doi.org/10.1145/321172.321179
19. Frick, M., Grohe, M.: The complexity of first-order and monadic second-order logic revisited. Ann. Pure Appl. Logic **130**(1-3), 3–31 (2004). https://doi.org/10.1016/j.apal.2004.01.007
20. Kupferman, O., Piterman, N., Vardi, M.Y.: Model Checking Linear Properties of Prefix-Recognizable Systems. In: Computer Aided Verification (CAV 2002). LNCS, vol. 2404, pp. 371–385. Springer Berlin Heidelberg (2002). https://doi.org/10.1007/3-540-45657-0_31
21. Kupferman, O., Piterman, N., Vardi, M.Y.: Pushdown Specifications. In: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2002). LNCS, vol. 2514, pp. 262–277. Springer Berlin Heidelberg (oct 2002). https://doi.org/10.1007/3-540-36078-6_18
22. Lautemann, C., Schwentick, T., Thérien, D.: Logics for context-free languages. In: Computer Science Logic, 8th International Workshop, CSL '94. pp. 205–216 (1994). https://doi.org/10.1007/BFb0022257
23. 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
24. Marx, M.: Conditional XPath. ACM Transactions on Database Systems **30**(4), 929–959 (dec 2005). https://doi.org/10.1145/1114244.1114247
25. McNaughton, R.: Parenthesis Grammars. JACM **14**(3), 490–500 (1967). https://doi.org/10.1145/321406.321411
26. McNaughton, R., Papert, S.: Counter-free Automata. MIT Press, Cambridge, USA (1971)
27. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (SFCS 1977). pp. 46–57 (Oct 1977). https://doi.org/10.1109/SFCS.1977.32
28. Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logics. J. ACM **32**(3), 733–749 (Jul 1985). https://doi.org/10.1145/3828.3837