Form Methods Syst Des (2007) 31: 265–279
DOI 10.1007/s10703-007-0040-7

The word problem for visibly pushdown languages described by grammars

Salvatore La Torre · Margherita Napoli · Mimmo Parente

Published online: 11 October 2007
© Springer Science+Business Media, LLC 2007

Abstract Visibly pushdown languages are an interesting subclass of deterministic context-free languages that can model nonregular properties of interest in program analysis. Such class properly contains typical classes of parenthesized languages such as "parenthesis", "bracketed", "balanced" and "input-driven" languages. It is closed under boolean operations and has decidable decision problems such as emptiness, inclusion and universality. We study the membership problem for visibly pushdown languages, and show that it can be solved in time linear in both the size of the input grammar and the length of the input word. The algorithm relies on a reduction to the reachability problem for game graphs. We also discuss the time complexity of the membership problem for the class of balanced languages which is the largest among those cited above. Besides the intrinsic theoretical interest, we further motivate our main result showing an application to the validation of XML documents against Schema and Document Type Definitions (DTDs).

Keywords Visibly pushdown grammars · Verification · XML

1 Introduction

Context-free languages are a very interesting class of languages that have been intensively studied by many researchers from different areas. Via their recursive characterization (the context-free grammars) they have played a central role in the development of compiler technologies, and recently, they have been also used to describe document formats over the

Work partially supported by funds for the research from MIUR 2006, grant "Metodi Formali per la verifica di sistemi chiusi ed aperti", Università di Salerno.

A preliminary version of this paper was published in the Proceedings of the 4th International Symposium Automated Technology for Verification and Analysis (ATVA 2006), Lecture Notes in Computer Science 4218, pp. 96–109, 2006.

S. La Torre (✉) · M. Napoli · M. Parente
Dipartimento di Informatica ed Applicazioni, Università degli Studi di Salerno,
Via Ponte Don Melillo, 84084 Fisciano (SA), Italy
e-mail: slatorre@unisa.it

Web (XML schema and Document Type Definitions) [16]. The automaton-like characterization of this class of formal languages, the pushdown automata, is a natural model for the control flow of sequential programs of typical procedural programming languages. Thus, program analysis, compiler optimizations, and program verification can be rephrased as decision problems for pushdown automata. As sample references on these topics see [2, 6, 9, 10, 12, 15, 23].

Often, the relevance of context-free languages cannot be fully exploited due to the intractability of many fundamental problems. In a recent paper Alur and Madhusudan [4] have introduced the class of *visibly pushdown languages* (VPLs). Visibly pushdown languages are context-free languages accepted by pushdown automata in which the input symbols determine the stack operations. They have been also characterized by the so-called *visibly pushdown grammars* (VPGs) [4]. This class of languages is rich enough to model nonregular properties and is also tractable and robust as the class of regular languages. In fact, VPLs are closed under all the boolean operations and some decision problems, such as inclusion and universality, are EXPTIME-complete while they are in general undecidable for the context-free languages. In [3], syntactic congruences on words and the problem of finding a minimal canonical deterministic pushdown automaton for VPLs are studied.

In this paper, we focus on the *membership problem*: "given a word w and a VPL language L, is $w \in L$?" While w is represented explicitly, L may be represented by either an automaton or a grammar and this leads to different approaches and hence different complexities. When a VPL language is represented by a VPA A, a simple algorithm for testing membership can be obtained by determinizing A by the construction given in [4] and then running the deterministic automaton on the input word. Clearly, it is not needed to compute the whole deterministic automaton (that would require exponential time) but the determinization can be carried out directly (on-the-fly) getting an $O(|Q|^2 \cdot |\Gamma| + |Q|^3)$ time upper bound, where Q is the set of states and Γ the set stack symbols of the nondeterministic pushdown automaton accepting the input language.

On the other hand, it is interesting to have an efficient algorithm also when the language is given by a VPG. It is known that for context-free languages, represented by grammars in Chomsky Normal Form, an efficient algorithm is the CYK algorithm. This algorithm runs in time cubic in the size of the word and linear in the size of the grammar. Time complexity improves on to quadratic if the grammar is not ambiguous [16]. More efficient algorithms have been given for particular subclasses of unambiguous grammars [1].

Here we give a solution to the membership problem for VPLs that takes time linear in both the size of the input word w and the size of the input visibly pushdown grammar G. Non-null productions of this kind of grammars are either of the form $X \to aY$ or of the form $X \to aYbZ$ where X, Y, Z are variables, a, b are terminal symbols, and in the latter production a and b correspond respectively to a push and its matching pop of the stack. The main idea of our algorithm is to reduce this problem to a two-player game H where: a player (the existential player) claims that she can show a derivation for a word and gives the next step in her proof; the other player (the universal player) challenges her to proceed in her proof on a portion of the remaining part of the word. Note that when a production of the form $X \to aY$ is picked by the existential player as next step in the proof of av, then the only claim on which the universal player can challenge her is on generating v from Y. Instead, when a production of the form $X \to aYbZ$ is picked as next step in the proof of $avbz$, then the universal player can challenge the existential player both on generating v from Y and z from Z. Clearly, the proof is completed when the existential player is asked to show that the empty word is generated from a variable X such that $X \to \varepsilon$ is a production of G. Therefore, if S is the start variable of G, we have that $w \in L(G)$ if and only if the existential player can prove the claim $S \Rightarrow^* w$ independently from the objections of the universal player.

The above game can be modeled as a reachability problem on a game graph of size linear in the size of w and G. Therefore, our result follows from the fact that the reachability problem on a game graph can be solved in linear time with essentially a depth-first search (cf. [24]).

We further show that the class of *balanced* languages introduced in [7], which extends the well-known *parentheses* and *bracketed* languages, is strictly contained in the class of VPLs, and that a balanced grammar can be translated in linear time to a language equivalent visibly push-down grammar. Therefore, we derive a linear time algorithm for the membership problem for all such class of languages.

Classical applications of the membership problem for formal languages have concerned with the parsing of programs and thus is strictly related to the design of compilers. In such a context, a language generator/acceptor is constructed once and for all and then it is used to parse several programs. Therefore, the size of the grammar or of the automaton can be considered constant and the efficiency of the algorithms is measured in terms of the length of the document (i.e., word) to parse. Also, the possibility of computing a deterministic model that captures the languages of interest guarantees efficient parsing independently of the complexity of the determinization procedure (determinization is done only once). Clearly, such observations do not apply when the language acceptor/generator may vary and this is the case of the type conformity checking of XML documents [26, 27]. In this paper, we show that the syntactic structure of a Document Type Definition (DTD) and of an XML schema can be efficiently captured by a visibly pushdown grammar, and thus our algorithm solving the membership problem for visibly pushdown languages can be used for efficiently checking the type of XML documents.

Finally, we discuss the complexity of translating visibly pushdown grammars to language equivalent visibly pushdown automata and vice-versa, and compare the algorithms to solving the membership problem that can be obtained using such translations with the direct ones. We also consider other algorithms that can be derived by standard approaches.

Organization of the paper In the next section we recall some definitions and introduce the notation we use in the rest of the paper. We provide our solutions to the membership problem for VPLs in Sect. 3 and for the class of balanced languages in Sect. 4. In Sect. 5, we give an application of our result that can be used to type-checking XML documents. In Sect. 6, we report a thorough discussion on the time complexity of other approaches. Finally, we conclude the paper with few remarks.

2 Preliminaries

In this section, we recall the definitions and introduce the notation we use in the rest of the paper.

2.1 Visibly pushdown languages

A *pushdown alphabet* is a tuple $\tilde{\Sigma} = \langle \Sigma_c, \Sigma_r, \Sigma_\ell \rangle$ consisting of three disjoint alphabets: Σ_c is a finite set of *calls*, Σ_r is a finite set of *returns* and Σ_ℓ is a finite set of *local actions*.

Visibly pushdown languages are characterized by a context-free grammar.

Definition 1 (Visibly pushdown grammar) A context-free grammar $G = (V, S, P)$, over an alphabet Σ, is a Visibly Pushdown Grammar (VPG) with respect to the pushdown alphabet $\tilde{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_\ell)$, if the set V of variables is partitioned into two disjoint sets $V0$ and $V1$, and the productions in P are of the following forms:

- $X \to \varepsilon$;
- $X \to aY$ such that if $X \in V0$ then $a \in \Sigma_\ell$ and $Y \in V0$;
- $X \to aYbZ$ such that $a \in \Sigma_c$ and $b \in \Sigma_r$ and $Y \in V0$ and if $X \in V0$ then $Z \in V0$.

Definition 2 A language of finite words $L \subseteq \Sigma^*$ is a visibly pushdown language (VPL) with respect to $\tilde{\Sigma}$ if there is a VPG G over $\tilde{\Sigma}$ such that $L(G) = L$.

A word w is *well-matched* if either $w \in \Sigma_\ell^*$ or $w = xaybz$ where x, y, z are well-matched, $a \in \Sigma_c$ and $b \in \Sigma_r$. In a word $w = uaxbv$, where $u, v, x \in \Sigma^*$ and x is well-matched, $a \in \Sigma_c$ and $b \in \Sigma_r$ are called *matching symbols*.[1] Observe that in any word w, for each *call* symbol there is at most one matching *return* symbol and vice-versa. Moreover, such matching relation can be efficiently determined in $O(|w|)$ time by parsing the input word once.

Directly from the definition, we can prove that from variables in $V0$ only well-matched words can be derived, while words that can be derived from variables in $V1$ are not necessarily well-matched.

Example 1 Consider the grammar $G = (V, S, P)$ over $\tilde{\Sigma} = \langle\{a\}, \{b\}, \{d\}\rangle$ where $V0 = \{X, Y\}$, $V1 = \{S\}$ and P has the following rules:

$$S \to \varepsilon | aS | bS | aXbS; \qquad X \to \varepsilon | aYbY; \qquad Y \to \varepsilon | dY.$$

It is easy to see that the word $w = a^3bdba \in L(G)$. Note that in w the first and the last occurrences of a are *unmatched*, while the others match with the b's. Note also that G is ambiguous (consider for example the word ab).

2.2 Game graphs

A *game graph* is a graph $H = (N, E)$ where N is a finite set of nodes partitioned into two sets $N_\exists$ and $N_\forall$, and $E \subseteq N \times N$ is the set of edges. A node of $N_\exists$ is called an $\exists$-*node* and a node of $N_\forall$ is called a $\forall$-*node*. A *strategy tree* from a node n_0 of H is a finite labeled tree obtained from H as follows. The root is labeled with n_0, and for each *internal* node u of the tree: if u is labeled with an $\exists$-node n_1 of H, then it has only a child which is labeled with a node n_2 such that $(n_1, n_2) \in E$; if u is labeled with a $\forall$-node n_1 of H, then for each n_2 such that $(n_1, n_2) \in E$ it has a child that is labeled with n_2. Given a game graph H, a starting node n_0 and a set T of nodes of H (called the *target set*), the reachability problem in H consists of determining if there exists a strategy tree from n_0 whose leaves are all labeled with nodes of T. We call such a strategy tree a *winning strategy*.

In Fig. 1, a simple game graph is shown. We have used a circle to denote an $\exists$-node and a box to denote a $\forall$-node. If we consider as target set $T = \{n_6\}$ then there are no strategy trees from n_1 whose leaves are all in T, that is the reachability problem is not satisfied. If instead we set $T = \{n_2, n_4\}$, the reachability problem is satisfied.

Reachability in game graphs can be solved at the cost of a depth-first search of the graph. Therefore, we have the following theorem, where $|H| = |N| + |E|$.

Theorem 1 *The reachability problem on a game graph H can be solved in $O(|H|)$ time.*

[1] Let us underline that we use *matching symbols* as a shorthand for *symbols in matching positions*.

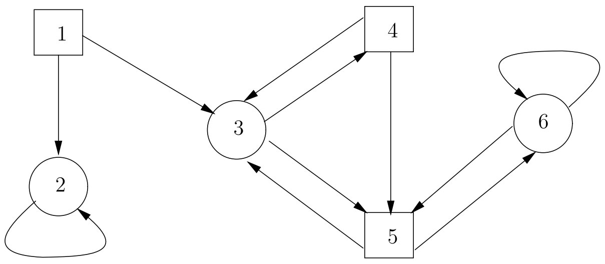

Fig. 1 An example of a game graph, where the ∃-nodes are circles and the ∀-nodes are boxes

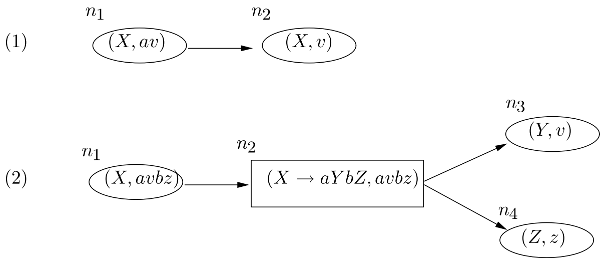

Fig. 2 Graphical representation of the construction rules of the game graph H_w^G

3 Membership problem

Given a string w and a language L, the *membership problem* consists of establishing whether w is in L. While w is represented explicitly, L can be represented by an automaton, a grammar, or an expression. For each of these representations a different algorithm, hence different time and space complexities, corresponds.

In this section we present an algorithm to decide whether a word $w \in \Sigma^*$ belongs to a language generated by a given VPG $G = (V, S, P)$. The main idea of this algorithm is to reduce our membership problem to reachability in game graphs. We assume that we have precomputed the matching symbols occurring in w.

The construction of the game graph Let G be a VPG (V, S, P) over $\tilde{\Sigma} = \langle \Sigma_c, \Sigma_r, \Sigma_\ell \rangle$ and $w \in \Sigma^*$. Define $H_w^G = (N, E)$ with $N = N_\forall \cup N_\exists$ and $N_\exists \subset (V \times \Sigma^*)$ and $N_\forall \subset (P \times \Sigma^*)$. The sets N and E are defined inductively as follows:

Let $(S, w) \in N_\exists$. Consider a node $n_1 = (X, u)$ in $N_\exists$, then

1. if $u = av$ and $(X \to aY) \in P$, then $n_2 = (Y, v) \in N_\exists$ and the edge $(n_1, n_2) \in E$ (see part (1) of Fig. 2);
2. if $u = avbz$ such that $a \in \Sigma_c$ and $b \in \Sigma_r$ are matching symbols and $(X \to aYbZ) \in P$, then
   - $n_2 = (X \to aYbZ, u) \in N_\forall$ and $(n_1, n_2) \in E$,
   - $n_3 = (Y, v) \in N_\exists$ and $(n_2, n_3) \in E$,
   - $n_4 = (Z, z) \in N_\exists$ and $(n_2, n_4) \in E$.
     (See part (2) of Fig. 2.)

The target set T consists of the nodes (X, ε), such that $X \to \varepsilon$ is a rule in P. Note that the graph H_w^G is a directed acyclic graph, having just one node with no incoming edges and such that the nodes in T do not have outgoing edges. Moreover each ∀-node has only one incoming edge (stemming from an ∃-node) and at most two outgoing edges (going into ∃-nodes).

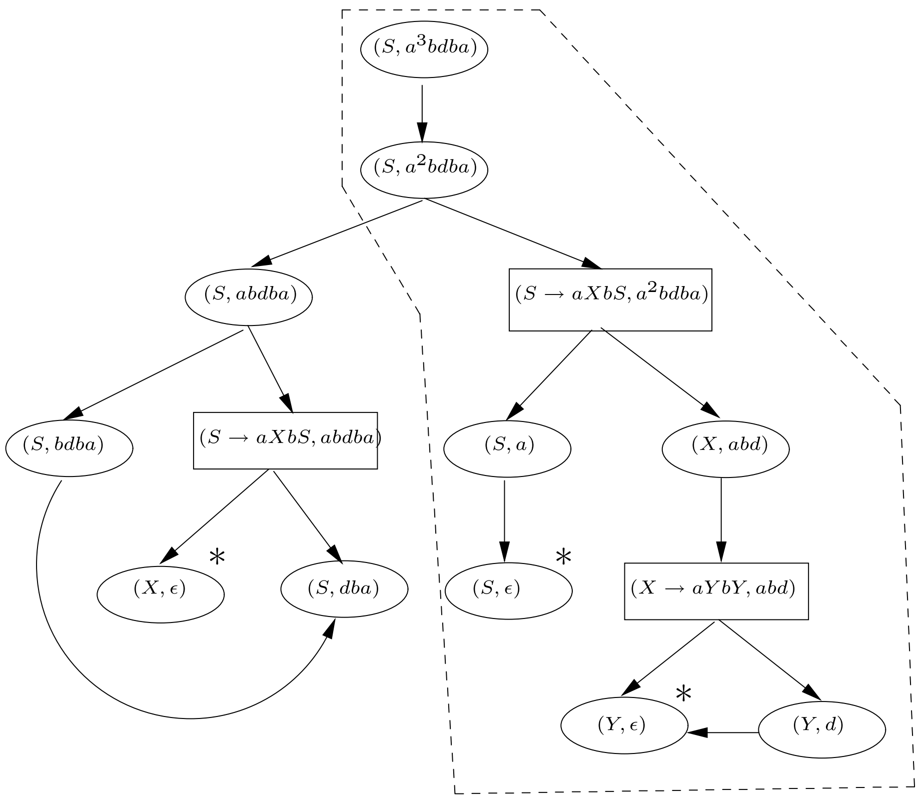

Fig. 3 The ∃-nodes are denoted with *circles* and the ∀-nodes with *boxes*. The nodes belonging to the target-set are starred. The subgraph enclosed in the dashed area corresponds to a winning strategy

Example 2 Given G as in the Example 1 and $w = a^3bdba$ the corresponding graph H_w^G is given in Fig. 3. It is immediate to see that $w \in L(G)$ and there is a strategy tree from (S, w), whose leaves are all in the target set (dashed area of Fig. 3).

In the above construction, it is clear that the words denoting the second component of the ∃-nodes and ∀-nodes are all subwords of w. Since the number of subwords of w is $O(|w|^2)$, we get an $O(|w|^2 \cdot |V|)$ upper bound on the number of ∃-nodes and an $O(|w|^2 \cdot |P|)$ upper bound on the number of ∀-nodes. Indeed, we can show a tighter upper bound on the number of the nodes in the graph, that is $O(|w| \cdot |P|)$. To this aim, in the next lemma we give a careful characterization of the form of the subwords effectively used in the construction of H_w^G.

Lemma 1 *If* $(Y, v) \in N_\exists$ *then*

1. *either* v *is a suffix of* w
2. *or* $Y \in V0$, v *is well-matched and there is a subword* w' *of* w *such that*
   2.1 *either* $w' = a_1\alpha v b_1$, *where* a_1, b_1 *are pairs of matching symbols and* $\alpha \in \Sigma_\ell^*$
   2.2 *or* $w' = a_1x_1a_2x_2b_2\alpha v b_1$, *where* a_1, b_1 *and* a_2, b_2 *are matching symbols*, $x_1, x_2 \in \Sigma^*$ *and* $\alpha \in \Sigma_\ell^*$.

Proof The proof is by structural induction on the definition of $N_\exists$. If (Y, v) is (S, w) then clearly v is a suffix of w. Suppose now that (Y, v) has an incoming edge then three cases can occur:

(i) The incoming edge comes from a $\exists$-node (X, u), and this implies that there is a production $X \to aY \in P$ and $u = av$. If u is a suffix of w then v is a suffix as well and case 1 holds. Otherwise, from the inductive hypothesis, u is well-matched and $X \in V0$ and $Y \in V0$. Thus, from the definition of G, $a \in \Sigma_\ell$. Moreover, there exists a subword w' of w such that either $w' = a_1 \alpha u b_1 = a_1 \alpha a v b_1$ or $w' = a_1 x_1 a_2 x_2 b_2 \alpha u b_1 = a_1 x_1 a_2 x_2 b_2 \alpha a v b_1$, where $\alpha a \in \Sigma_\ell^*$ and thus 2 holds for (Y, v).

(ii) The incoming edge comes from a $\forall$-node $(X \to aYbZ, u)$ and in turn this has an incoming edge from the $\exists$-node (X, u) where $u = avbz$, for some $z \in \Sigma^*$. Since a, b are matching symbols, then $Y \in V0$ and v is well-matched. From the inductive hypothesis u is a subword of w and then $w' = avb$ is a subword of w as well (case 2.1 holds).

(iii) The incoming edge comes from a $\forall$-node $(X \to aZbY, u)$ and this in turn has an incoming edge from the $\exists$-node (X, u) where $u = azbv$, for some $z \in \Sigma^*$. If u is a suffix of w then v is a suffix of w as well and thus case 1 holds. Otherwise, u is well-matched and there exists a subword w' of w such that either $w' = a_1 \alpha u b_1 = a_1 \alpha a z b v b_1$ or $w' = a_1 x_1 a_2 x_2 b_2 \alpha u b_1$ which can be written as $a_1 x' a z b v b_1$ where $x' = x_1 a_2 x_2 b_2 \alpha$. Moreover $X \in V0$ implies that $Y \in V0$ too and thus case 2 holds. □

Note that the second component of a $\forall$-node is identical to the second component of the $\exists$-node which precedes it.

A simple characterization of the words in the second component of the nodes of the game graph H_w^G can be obtained using the following definition. A well-matched subword v of w is called *right maximal* if $w = uvx$ and, for each prefix $x' \neq \varepsilon$ of x, the subword vx' is not well-matched. In fact, from Lemma 1, if v is not a suffix of w, it is well matched and is immediately followed by a return symbol whose matching call precedes v. Therefore, v is right maximal, and the following corollary holds.

Corollary 1 *If $(X, v) \in N_\exists$ then v is either a suffix of w or a right maximal well-matched subword of w.*

In the next lemma we state an upper bound on the number of different subwords of w which appear in the nodes of H_w^G.

Lemma 2 *The number of different words u such that $(X, u) \in N_\exists$ is $O(|w|)$.*

Proof Fix $w = w_1 \dots w_n$ and denote by $w(i, j)$ the subword of w from the i-th through the j-th position (i.e., $w(i, j) = w_i \dots w_j$). From Corollary 1, we have that if $(Y, v) \in N_\exists$ then v is either a suffix of w or a right maximal well-matched subword of w. Thus, for $i = 1, \dots, n$, if $v = w(i, h)$ for some $i \le h \le n$ and $(Y, v) \in N_\exists$ then one of the following cases holds:

- w_i is either a matched call or a local action: then v is either $w(i, n)$ or the right maximal well-matched subword starting at position i;
- w_i is either an unmatched call or a return: then $v = w(i, n)$.

Observe that once a position i is fixed, there is just one right maximal well-matched subword starting at position i. Therefore, the number of the subwords u of w such that there exists a $(X, u) \in N_\exists$ is bounded by $(2|w| - n_{uc} - n_r)$, where n_{uc} is the number of unmatched calls in w and n_r is the number of returns. □

We can now compute the size of H_w^G.

Lemma 3 *The size of the graph H_w^G is $O(|w| \cdot |G|)$. Moreover, H_w^G can be constructed in $O(|w| \cdot |G|)$ time.*

Proof From Lemma 2 and the definition of H_w^G, the number of $\exists$-nodes is bounded by $O(|w| \cdot |V|)$. Since the second component of a $\forall$-node is identical to the second component of the $\exists$-node which precedes it, we have that the number of $\forall$-nodes is $O(|w| \cdot |P|)$. Moreover, observe that the number of out-going edges from each $\exists$-node (X, v) is bounded by the number of G productions rewriting variable X and there are only two out-going edges from each $\forall$-node. Thus also the number of edges is $O(|w| \cdot |P|)$. Therefore the first part of the lemma follows.

To show that H_w^G can be constructed in $O(|w| \cdot |G|)$ time, observe that for each variable X and for each $\exists$-node (X, u), one has just to go once through all the productions of the form $X \to \alpha$ (the $\forall$-nodes and the edges are uniquely determined by such productions and the $\exists$-nodes). Using appropriate data structures this can be done in time proportional to the number of such productions. Thus, adding up over all the possible words u and variables X, by Lemma 2 we get that the total time to construct H_w^G is $O(|w| \cdot |G|)$. □

In the next Lemma we prove that a strategy tree from (S, w) to nodes of the target set exists if and only if $w \in L(G)$ (cf. Example 2).

Lemma 4 *For every $(X, u) \in N_\exists$ there is a strategy tree from (X, u) to the target set T if and only if there is a derivation from X to u in G.*

Proof We prove the assert by induction on the definition of $N_\exists$, using the nodes with no outgoing edges as base case. Let (X, u) be one of these nodes. If $u = \varepsilon$ then clearly a derivation from X to u exists if and only if $(X, u) \in T$. If $u \neq \varepsilon$ then $(X, u) \notin T$. As (X, u) has no outgoing edges, there are no productions of G from X that can start a derivation to u. Thus, a derivation from X to u does not exist. For the induction step we prove first the *only-if* part and let e be the outgoing edge branching off the root (X, u) of the strategy tree. If e is added in the H_w^G construction for a production of the form $X \to aY$, then an $\exists$-node (Y, v) exists having e as an incoming edge and $u = av$. Then by induction hypothesis there is a derivation from Y to v, and thus a derivation from X to u. If, on the other hand, e is added for a production of the form $X \to aYbZ$, then a $\forall$-node having e as an incoming edge exists. In H_w^G, each $\forall$-node is followed by two $\exists$-nodes within a strategy tree, let (Y, v) and (Z, z), with $u = avbz$, be the labels of such nodes. By induction, $Y \Rightarrow^* v$ and $Z \Rightarrow^* z$ holds. Thus, the following derivation exists: $X \Rightarrow aYbZ \Rightarrow^* avbZ \Rightarrow^* avbz$.

The *if* part can be easily proved analogously, by constructing the strategy tree from the derivation of u from X. □

Now from Lemmas 3 and 4 our main result follows.

Theorem 2 *The membership problem for a* VPG *$G = (V, S, P)$ over Σ and a word $w \in \Sigma^*$ is decidable in $O(|w| \cdot |G|)$.*

4 Membership for grammars of well-matched words

Several kinds of grammars generating subsets of Dyck languages have been studied in the past and are strictly related to VPG grammars. In this section we briefly describe some of these grammars and prove that the classes of generated languages are strictly contained in VPL. From this result we obtain an upper bound on the complexity of the membership problem for all of them.

The best known examples of grammars generating well-matched words are the parenthesis grammars, defined by Mc Naughton in 1967 [20] and the bracketed grammars, introduced by Ginsburg and Harrison [14]. A *parenthesis* grammar is a context-free grammar with set of variables V and alphabet $\Sigma \cup \{(,)\}$, where each rule is of the form $X \rightarrow (\alpha)$, $\alpha \in (\Sigma \cup V)^*$. One of the most relevant results for this class was obtained by Knuth [17] who showed the existence of an algorithm for determining whether a context-free language admits a parenthesis grammar (actually the class of languages considered by Knuth is slightly larger than that defined by Mc Naughton since a word in a language does not needed to be surrounded by parenthesis).

A *bracketed* grammar differs from a parenthesis grammar because of a set of indexed parentheses and a bijection between parentheses and production rules. In fact, any rule i of a bracketed grammar is of the form $X \rightarrow (_i\alpha)_i$, $\alpha \in (\Sigma \cup V)*$, and $(_i \neq (_j$ for $i \neq j$.

More recently, the class of so-called *balanced* grammars has been introduced [7] and many interesting properties of balanced languages have been studied. Balanced grammars extend both parenthesis and bracketed grammars. In these grammars the set of productions for each variable is a regular set (as in the XML grammars, studied in [8] and considered in Sect. 5). More precisely, let A be a set of open parentheses, $\bar{A} = \{\bar{a} \mid a \in A\}$ be the set of closed parentheses and Σ be an alphabet. A balanced grammar is defined as follows.

Definition 3 (Balanced grammar) A balanced grammar over an alphabet $A \cup \bar{A} \cup \Sigma$ is a tuple $G = (V, S, P, R)$ such that:

- V is the set of variables and $S \in V$ is the axiom;
- $R = \{R_{X,a} \mid X \in V, a \in A\}$, for regular sets $R_{X,a}$ over the alphabet $V \cup \Sigma$;
- P is the set of productions of the form $X \rightarrow a\alpha\bar{a}$ where $X \in V, a \in A$ and α belongs to $R_{X,a}$.

The languages defined by all the above described grammars are deterministic context-free languages and the membership problem can thus be solved in time which is linear with respect to the length of the input word. Moreover, Lynch [19] studied the membership problem for parenthesis languages and showed that it is in LOGSPACE with respect to the size of the input word. Note that the complexity of this problem with respect to the size of the grammar has not been addressed. In [11, 22, 25], the class of *input-driven* languages has been introduced, which coincides with well matched VPLs. In these papers the space complexity of the membership problem is analyzed when the languages are given as automata (instead of grammars).

Here we determine the complexity of the membership problem with respect to the size of the grammar (besides the size input word). For some of the considered kinds of grammar, easy algorithms to check the membership (without modifying the grammar) can be given. For instance, the bijection between parentheses and production rules in bracketed grammars allows to uniquely determine, at each step of a derivation for a given word, the production which corresponds to the parenthesis under reading, thus checking the membership. On the other side, the problem is more interesting for the balanced grammars and we solve it by showing the containment of balanced languages in the class of VPLs, where call and return symbols play the role of open and closed parentheses. Let us observe that the containment is strict since VPLs are closed under concatenation [4] while balanced languages are not (actually no word in a balanced language is the concatenation of two non-empty well matched words). Moreover, observe that there are also other differences between balanced languages and VPLs. First, balanced languages are defined over a more restrict kind of alphabet, with a one-to-one correspondence between open and closed parentheses and, further, all the words in a balanced language are well matched, while VPLs contain also non well-matched words.

In the following lemma we show that a balanced grammar can be translated into a VPL grammar whose size is linear in the size of the balanced grammar and the number of open parentheses. Observe that the set of productions of a balanced grammar may be infinite, thus the size of the grammar is defined in terms of the size of the representation of the regular languages $R_{X,a}$. Here we assume that they are represented by right-linear grammars.[2] Thus the size of a Balanced Grammar (V, S, P, R) is $|P| + \sum_{X\in V, a\in A} |P_{X,a}|$, where $P_{X,a}$ is the set of the productions of a right-linear grammar generating $R_{X,a}$. Observe that this grammar has the same size as a nondeterministic finite automaton recognizing $R_{X,a}$.

Lemma 5 *Given a balanced grammar G over an alphabet $A \cup \bar{A} \cup \Sigma$ there exists a visibly pushdown grammar G' with respect to $\tilde{\Sigma} = \langle A, \bar{A}, \Sigma\rangle$ such that $L(G') = L(G)$ and $|G'| = O(|G| \cdot |A|)$.*

Proof Let G be the balanced grammar (V, S, P, R). For each regular set $R_{X,a}$ in R, let $G_{X,a} = (V_{X,a}, S_{X,a}, P_{X,a})$ be a right-linear grammar over $V \cup \Sigma$ such that $L(G_{X,a}) = R_{X,a}$. We assume that for each $X \in V$ and $a \in A$ the sets $V_{X,a}$ are pairwise disjoint. Denote $V' = \bigcup_{\substack{X\in V\\ a\in A}} V_{X,a}$, and $P' = \bigcup_{\substack{X\in V\\ a\in A}} P_{X,a}$.

From G we can easily obtain a language equivalent context-free grammar by simply “linking” the grammars $G_{X,a}$ to the productions of G that place parenthesis out of the words of $R_{X,a}$. More precisely, we need to replace all the rules of the form $X \to a\alpha\bar{a}$ of G ($\alpha \in R_{X,a}$) with the sole rule $X \to aS_{X,a}\bar{a}$, and then take the union of all the productions of grammars $G_{X,a}$ (i.e., P'). Denote with G'' the resulting grammar. Clearly, $L(G'') = L(G)$, $|G''| = O(|G|)$ and the productions of G'' are of the forms: $X \to aS_{X,a}\bar{a}$, $Y \to XZ$, $Y \to cZ$ and $Y \to \varepsilon$ (we have denoted with X a variable from V, with Y, Z variables from V', with a a parenthesis in A and with c a symbol in Σ).

Because of the productions of the forms $X \to aS_{X,a}\bar{a}$ and $Y \to XZ$, grammar G'' might not be a VPG. Therefore we define a new grammar G' where those productions are replaced. In particular, each production of G'' rewriting the axiom, i.e., of the form $S \to aS_{S,a}\bar{a}$, is replaced with $S \to aS_{S,a}\bar{a}\Lambda$ where Λ is a new variable from which only the empty word can be derived. Each production of the form $Y \to XZ$ is combined with productions of the form $X \to aS_{X,a}\bar{a}$, thus obtaining productions of the form $Y \to aS_{X,a}\bar{a}Z$. It is simple to verify that such modifications of the grammar G'' do not alter the generated language, and the resulting grammar G' is visibly pushdown. Also, all the variables of grammar G, besides the axiom, are not used in the resulting productions.

Formally, define the pushdown alphabet $\tilde{\Sigma}$ as follows: A is the set of calls, $\bar{A}$ is the set of returns and Σ is the set of local actions. The variable set of G' is $\{S, \Lambda\} \cup V'$ (where Λ is

[2] Here we consider right-linear grammars whose productions are of the following form: $X \to xY$ or $X \to \epsilon$, for variables X, Y and terminals x.

a new symbol). The variables of G' derive only well-matched words (i.e., are all of the kind $V0$). The axiom is S (the axiom of G). The set of productions is the smallest set containing:

- a production $S \to aS_{S,a}\bar{a}\Lambda$, for each $a \in A$ such that $R_{S,a}$ is not empty;
- a production $\Lambda \to \varepsilon$;
- a production $Z \to aS_{X,a}\bar{a}Y$ for each production $Z \to XY$ in P', $X \in V$, $Z, Y \in V'$, and for each open parenthesis a such that $R_{X,a}$ is not empty;
- all productions of the form $Z \to cY$ and $Z \to \varepsilon$ of P', for $c \in \Sigma$.

For the above observations we get $L(G') = L(G)$. Moreover, the number of G' variables is $2 + |V'|$, and the number of G' productions is at most $|A| \cdot |P'|$. Therefore, we obtain the claimed bound. $\square$

From Lemma 5 and Theorem 2, we have the following corollary.

Corollary 2 *The membership problem for a word w and a balanced grammar G, over an alphabet $A \cup \bar{A} \cup \Sigma$, is decidable in $O(|w| \cdot |G| \cdot |A|)$.*

5 An application: validation of XML documents

In this section, we describe an interesting application of our membership problem for VPLs to the validation of XML documents. We start giving the definition of what we call XML-schema grammars which captures the syntactic structure of an *XML schema*. We use XML schemas as they include the full capabilities of *Document Type Definitions* (DTDs) and in addition they can define custom data types (besides primitive ones) using object-oriented data modeling principles: encapsulation, inheritance, and substitution (in [8] a similar definition for the DTD was given).

Let Σ be a finite alphabet, we define by $\bar{\Sigma}$ the set of symbols $\bar{\sigma}$ such that $\sigma \in \Sigma$. A symbol $\sigma \in \Sigma$ denotes an "opening tag" and $\bar{\sigma}$ its matching "closing tag". In the XML schema each tag has associated to itself a finite number of types (called also refinements or specifications, of a given base type), let n_σ be such a number. Call $\hat{\Sigma} = \Sigma \cup \bar{\Sigma}$. *XML schema grammars* are balanced grammars where opening and closing tags correspond to open and closed parentheses (and there are no other terminal symbols). In XML schema Grammars, the set V of non-terminal symbols is partitioned into subsets, each of them corresponding to an opening tag.

Definition 4 (XML schema grammar) An XML schema grammar $G = (V, S, P, R)$, over an alphabet $\hat{\Sigma}$ is such that (n_σ indicating the different types for each open tag $\sigma \in \Sigma$):

- $V = \bigcup_{\substack{\sigma \in \Sigma \\ 1 \le i \le n_\sigma}} X_\sigma^i$ (each non-terminal symbol is in a one-to-one correspondence to an opening tag σ and a positive integer which is at most n_σ);
- $S \in V$ is the axiom;
- $R_\sigma^i, \sigma \in \Sigma$ and $1 \le i \le n_\sigma$, is a regular language over V, such that no word in it contains two different elements X_τ^i and X_τ^j, $i \ne j$, for any $\tau \in \Sigma$;[3]
- $R = \{R_\sigma^i \mid \sigma \in \Sigma, i = 1, \dots, n_\sigma\}$.

[3]This requirement is a requirement of W3C that practically forbids to use two or more different types, derived from the same base type tag, as siblings in the document.

- productions in P are of the form $X^i_\sigma \to \sigma\alpha\bar{\sigma}$ where $\alpha \in R^i_\sigma$. (Note that, the tag on the right-hand side of the production is uniquely determined by the non-terminal on the left-hand side.)

An *XML schema language* is the language generated by an XML schema grammar. In the following, we assume that each regular language is represented by a right-linear grammar that generates it. Let us underline that an XML schema grammar with just one type for each tag suffices to describe DTD's, cf. [8].

Any XML schema language can be seen as a visibly pushdown language where opening tags correspond to call symbols and closing tags to their matching return symbols. Thus, an XML schema language over $\hat{\Sigma}$ can be generated by a visibly pushdown grammar over the alphabet $\tilde{\Sigma} = (\Sigma, \bar{\Sigma}, \emptyset)$ as shown in the following lemma.

Lemma 6 *Given an XML schema grammar G_{XML} over an alphabet $\hat{\Sigma}$, there exists a visibly pushdown grammar G over $\tilde{\Sigma} = (\Sigma, \bar{\Sigma}, \emptyset)$ such that $L(G) = L(G_{\mathrm{XML}})$ and $|G| = O(|G_{\mathrm{XML}}|)$.*

Proof Let G_{XML} be the XML schema grammar $(V_{\mathrm{XML}}, S_{\mathrm{XML}}, P_{\mathrm{XML}}, R_{\mathrm{XML}})$. For each set R^i_σ, let $G^i_\sigma = (V^i_\sigma, S^i_\sigma, P^i_\sigma)$ be a regular right-linear grammar over V_{XML} such that $L(G^i_\sigma) = R^i_\sigma$. (We return on this assumption after completing the proof of the lemma.)

We construct a visibly pushdown grammar $G = (V, S, P)$ such that $L(G) = L(G_{\mathrm{XML}})$ as follows. The set $V = V0$ is $\{S, \Lambda\} \cup \bigcup_{\substack{\sigma\in\Sigma \\ 1\le i\le n_\sigma}} V^i_\sigma$ where S and Λ are new symbols. The set of productions P is the smallest set containing:

- a production $S \to \sigma S^i_\sigma \bar{\sigma} \Lambda$, if $S_{\mathrm{XML}} = \mathrm{x}^i_\sigma$ (i.e., S_{XML} is the non-terminal symbol corresponding to tag σ and type i);
- a production $\Lambda \to \varepsilon$;
- a production $Z \to \sigma S^j_\sigma \bar{\sigma} Y$ for each opening tag σ and $1 \le j \le n_\sigma$ and for each production $Z \to \mathrm{x}^j_\sigma Y$ of G^i_σ;
- a production $Z \to \varepsilon$ for each production $Z \to \varepsilon$ of G^i_σ, where $\sigma \in \Sigma$ and $1 \le i \le n_\sigma$.

The cardinality of V is $2 + \sum_{\substack{\sigma\in\Sigma \\ 1\le i\le n_\sigma}} |V_\sigma|$, and the number of productions in P is $2 + \sum_{\sigma\in\Sigma 1\le i\le n_\sigma} |P_\sigma|$. Therefore, we obtain the claimed bound. □

In the above proof, we have assumed that each regular language R^i_σ is given as a right-linear grammar. According to W3C recommendation [26], such regular languages are required to admit a *deterministic regular expression*, that is an expression for which the corresponding Glushkov finite automaton, see [13], is deterministic. In [5], it has been shown that this finite automaton can be constructed in time linear in the size of the deterministic regular expression.[4] Therefore, it is without loss of generality to assume, that the size of right-linear grammars G^i_σ generating languages R^i_σ have linear size in the size of the regular expression used in the XML schemas (or DTDs).

Given an XML document D and an XML schema, or equivalently a DTD, H, the *XML type-checking problem* is the problem of checking if D syntactically conforms to H. This

[4] A Glushkov automaton for a regular expression E is an automaton whose states correspond to the positions (occurrences) of symbols in E and whose transitions connect positions that can be consecutive on a path through E. Such automaton can be obtained with standard algorithms (independently given by McNaughton and Yamada [21] and Glushkov [13]).

problem can be formalized as a membership problem for XML languages. From the above lemma and Theorem 2, we have the following corollary.

Corollary 3 *The type-checking problem for XML documents is decidable in time linear in the length of the document and in the size of the XML schema* (*or DTD*).

6 Complexity of other approaches

In literature VPL languages have been represented also with *visibly pushdown automata* (VPA). In this section, we first recall the definition of such automata and some basic results. Then we briefly discuss on the computational time complexity of other solutions to the membership problem for VPLs considering as a starting representation either VPGs or VPAs. We use translations from one representation to the other or more trivial algorithms, without explicitly giving the proofs.

In a Visibly Pushdown Automaton over $\tilde{\Sigma}$, call, return and local symbols determine the stack operations.

Definition 5 (Visibly pushdown automata) A (nondeterministic) Visibly Pushdown Automaton (VPA) on finite words over $\langle \Sigma_c, \Sigma_r, \Sigma_\ell \rangle$ is a tuple $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ where Q is a finite set of states, $Q_{in} \subseteq Q$ is a set of initial states, Γ is a finite stack alphabet that contains a special bottom-of-stack symbol $\perp$, $\delta \subseteq (Q \times \Sigma_c \times Q \times (\Gamma \setminus \{\perp\})) \cup (Q \times \Sigma_r \times \Gamma \times Q) \cup (Q \times \Sigma_\ell \times Q)$, and $Q_F \subseteq Q$ is a set of final states.

Let us remark that the acceptance is only on final states (not by *empty-stack*) and ε-transitions are not allowed. The languages accepted by a VPA over $\tilde{\Sigma}$ is a context-free language over an alphabet $\Sigma = (\Sigma_c \cup \Sigma_r \cup \Sigma_\ell)$.

Now we recall some known results about VPL languages.

Theorem 3 [4] *For a pushdown alphabet $\tilde{\Sigma}$, a language L is* VPL *if and only if it can be accepted by a visibly pushdown automaton.*

Theorem 4 [4] *For any* VPA *M over $\tilde{\Sigma}$, there is a deterministic* VPA *M' over $\tilde{\Sigma}$ such that $L(M') = L(M)$. Moreover, if M has n states, we can construct M' with $O(2^{n^2})$ states and with stack alphabet of size $O(2^{n^2} \cdot |\Sigma_c|)$.*

In what follows, we fix a VPG $G = (V, S, P)$ over an alphabet Σ, a VPA $A = (Q, Q_{in}, \Gamma, \delta, Q_F)$ and a word $w = w_1 w_2 \dots w_n$, where $w_i \in \Sigma$ for $i = 1, \dots, n$ and $n \geq 0$.

When a VPL language is given as a VPA A, since this class of automata is determinizable (see Theorem 4), we can use the determinization construction on-the-fly while checking for language membership. From the construction given in [4], it is easy to see that this approach leads to an algorithm that takes time which is cubic in the number of states of the nondeterministic automaton: $O(|Q|^2 \cdot |\Gamma| + |Q|^3)$.

Alternatively, we could transform a VPA A into an equivalent VPG G_A and this, using again the construction given in [4], amounts to a set of rules P of size $|Q|^4 \cdot |\Gamma|^2 \cdot |\Sigma_c| \cdot |\Sigma_r|$.

Consider now the case when a VPL is represented by a VPG. A standard algorithm to solve membership for context-free languages is the well known CYK algorithm [1]. Starting from a grammar in Chomsky Normal Form and a word $w = w_1 \cdots w_n$, this algorithm consists of computing for each subword $w_i \cdots w_{i+j-1}$ the set of all variables from which it can be derived. The main step of the computation consists of adding up a variable X, to the set of variables for $w_i \cdots w_{i+j-1}$, whenever: (1) the grammar contains a production $X \to YZ$, and (2) there exists a k, $1 \leq k \leq j$, such that $w_i \cdots w_{i+k-1}$ can be derived form Y and $w_{i+k} \cdots w_{i+j-1}$ can be derived from Z. The algorithm can be implemented to run in $O(|P| \cdot n^3)$ time [1].

Since VPGs have productions with at most two variables on the right hand side, the CYK algorithm can be easily adapted to solve also the membership problem for them. From the definition of a VPG, we know that the splitting of a subword $w_i \cdots w_{i+j-1}$ according to a rule of the form $X \to aYbZ$ is uniquely determined. Also, rules of the form $X \to aY$ deterministically split the subword. Therefore, a simple adaptation of the CYK algorithm can be implemented in $O(|P| \cdot n^2)$ time. A further improvement can be achieved if we choose to compute the sets of variables only for the subwords that are actually determined in the parsing of the input word according to the rules of the grammar. According to the results shown in Sect. 3 the number of such subwords is $O(n)$, therefore the time complexity of the algorithm reduces to $O(|P| \cdot n)$, the same as for the solution we have proposed in this paper. It is worth to mention that an efficient way of determining such subwords is to mimic the construction of the game graph given in Sect. 3.

Alternatively, one can think of translating the VPG into an equivalent VPA, but this turns out to be quite expensive in general. In fact, using also here the construction given in [4], translating a VPG into a VPA costs $O(|P| + |P_\varepsilon| \cdot |\Sigma_r| \cdot |V|)$ time, where $|P_\varepsilon|$ is the subset of the *nullable* productions (productions of the form $X \to \varepsilon$). The size of the set of states and the size of the stack alphabet of the VPA are respectively $|V|$ and $|V \cdot \Sigma_r|$.

7 Conclusions

The membership problem is a central decision problem in the formal languages theory. The time complexity of the membership problem for subclasses of context-free languages has been largely studied, mainly because of its importance in parsing (see also [1]).

In this paper, we have addressed the membership problem for visibly pushdown languages, a sub-class of deterministic context-free languages. Using the visibly pushdown grammars from [4], we have given an algorithm to solve this problem in time linear in both the length of the input word and the size of the grammar. Thus, checking for membership in visibly pushdown grammars can be done faster than for general context-free grammars (even in the case of unambiguous grammars [16]). (Recall that the membership problem in regular languages is linear both in the size of the automaton/grammar and the length of the input word.) As for the other decision problems, the complexity of checking for membership confirms that visibly pushdown languages have nice features in terms of tractability and robustness, and thus from this point of view are more alike the class of regular languages than to the class of context-free languages. As shown in Sect. 5, our result on the membership problem for VPLs has a natural application in the processing of XML documents. The use of visibly push-down automata for solving problems for XML that involve processing of documents from left to right (such as the type-checking problem we have considered in this paper) has been recently proposed in [18]. There, the authors give an automaton counterpart to XML-grammars that they call XVPA (a variant of visibly push-down automata) and rephrase some typing and streaming problems for XML (including the type-checking problem) as automata decision problems.

References

1. Aho A, Ullman J (1973) The theory of parsing, translation and compiling, vol I. Prentice-Hall, Englewood Cliffs
2. Alur R, Benedikt M, Etessami K, Godefroid P, Reps TW, Yannakakis M (2005) Analysis of recursive state machines. ACM Trans Program Lang Syst 27(4):786–818
3. Alur R, Kumkar V, Madhusudan P, Viswanathan M (2005) Congruences for visibly pushdown languages. In: Proceedings of the 32nd international colloquium of automata, languages and programming (ICALP'05). Lecture notes in computer science, vol 3580. Springer, Berlin, pp 1102–1114
4. Alur R, Madhusudan P (2004) Visibly pushdown languages. www.cis.upenn.edu/~alur/. A preliminary version appears in: Proceedings of the 36th ACM symposium on theory of computing (STOC'04), pp 201–211
5. Brggemann-Klein A (1993) Regular expression into finite automaton. Theor Comput Sci 120(2):197–213
6. Ball T, Rajamani S (2000) Bebop: a symbolic model checker for boolean programs. In: Proceedings of the 7th international workshop on model checking of software (SPIN 2000). Lecture notes in computer science, vol 1885. Springer, Berlin, pp 113–130
7. Berstel J, Boasson L (2002) Balanced grammars and their languages. In: Formal & natural computing: essay dedicated to Grzegorz Rozenberg. Lecture notes in computer science, vol 2300. Springer, Berlin, pp 3–25
8. Berstel J, Boasson L (2002) Formal properties of XML grammars and languages. Acta Inform 38:649–671
9. Bouajjani A, Esparza J, Maler O (1997) Reachability analysis of pushdown automata: application to model-checking. In: Proceedings of the 8th international conference on concurrency theory (CONCUR'97). Lecture notes in computer science, vol 1243. Springer, Berlin, pp 135–150
10. Burkart O, Steffen B (1992) Model checking for context-free processes. In: Proceedings of the 3rd international conference on concurrency theory (CONCUR'92). Lecture notes in computer science, vol 620. Springer, Berlin, pp 123–137
11. Dymond PW (1988) Input-driven languages are recognized in log n space. Inf Process Lett 26:2472250
12. Esparza J, Kucera A, Schwoon S (2003) Model checking LTL with regular valuations for pushdown systems. Inform Comput 186(2):355–376
13. Glushkov VM (1961) The abstract theory of automata. Russ Math Surv 16:1–53
14. Ginsburg S, Harrison MA (1967) Bracketed context-free languages. J Comput Syst Sci 1:1–23
15. Henzinger TA, Jhala R, Majumdar R, Necula GC, Sutre G, Weimer W (2002) Temporal-safety proofs for systems code. In: Proceedings of the 14th international conference on computer-aided verification (CAV 2002). Lecture notes in computer science, vol 2404. Springer, Berlin, pp 526–538
16. Hopcroft JE, Motwani R, Ullman JD (2001) Introduction to automata theory, languages, and computation. Addison–Wesley, Reading
17. Knuth DE (1967) A characterization of parenthesis languages. Inf Control 11(3):269–289
18. Kumkar V, Madhusudan P, Viswanathan M (2006) Visibly pushdown languages for XML. Technical Report UIUCDCS-R-2006-2704, UIUC
19. Lynch N (1977) Log space recognition and translation of parenthesis languages. J ACM 24(4):583–590
20. McNaughton R (1967) Parenthesis grammars. J ACM 14(3):490–500
21. McNaughton R, Yamada H (1960) Regular expressions and state graphs for automata. IRE Trans Electron Comput 9:39–47
22. Mehlhorn K (1980) Pebbling mountain ranges and its application of DCFL-recognition. In: Proceeding of the 7th international colloquium of automata, languages and programming (ICALP'80). Lecture notes in computer science, vol 85. Springer, Berlin, pp 422–435
23. Reps TW, Horowitz S, Sagiv S (1995) Precise interprocedural dataflow analysis via graph reachability. In: Proceeding of the 22nd symposium on principles of programming languages (POPL'95), pp 49–61
24. Thomas W (1995) On the synthesis of strategies in infinite games. In: 12th annual symposium on theoretical aspects of computer science (STACS'95). Lecture notes in computer science, vol 900. Springer, Berlin, pp 1–13
25. von Braunmhl B, Verbeek R (1983) Input-driven languages are recognized in $\log n$ space. In: Proceeding of FCT. Lecture notes in computer science, vol 158. Springer, Berlin, p 4051
26. W3C recommendation (2000) Extensible markup language (XML) 1.0, 2nd edn. http://www.w3.org/TR/REC-xml. Accessed 6 October 2000
27. W3C recommendation (2001) XML schema part 0,1 and 2. http://www.w3.org/TR/xmlschema-0,1,2. Accessed 2 May 2001