Active Context-Free Games*

Anca Muscholl[1], Thomas Schwentick[2], and Luc Segoufin[3]

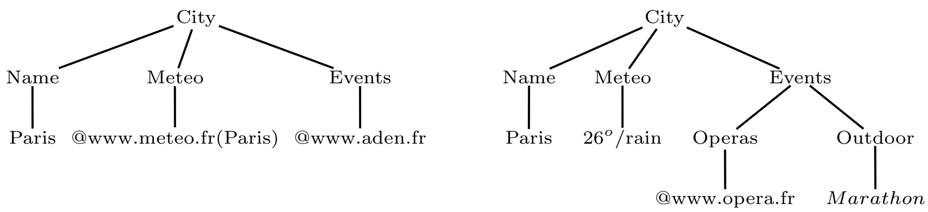
[1] LIAFA, Université Paris VII, 2 pl. Jussieu, F-75251 Paris
[2] Philipps-Universität Marburg, FB Mathematik und Informatik, D-35032 Marburg
[3] INRIA, Parc Club Orsay Univ., ZAC des vignes, 4 rue J. Monod, F-91893 Orsay

Abstract. An *Active* Context-Free Game is a game with two players (Romeo and Juliet) on strings over a finite alphabet. In each move, Juliet selects a position of the current word and Romeo rewrites the corresponding letter according to a rule of a context-free grammar. Juliet wins if a string of the regular target language is reached. We consider the complexity of deciding winning strategies for Juliet depending on properties of the grammar, of the target language, and on restrictions on the strategy.

1 Introduction

This work was motivated by implementation issues that arose while developing *active XML* (AXML) at INRIA. Active XML extends the framework of XML for describing semi-structured data by a dynamic component, allowing to cope with e.g. web services and peer-to-peer architectures. For an extensive overview of AXML we refer to [2, 3, 11].

We briefly describe here the background needed for understanding the motivation of this work. Roughly speaking, an AXML document consists of some explicitly defined data, together with some parts that are defined only intensionally, by means of embedded calls to web services [3, 9, 7]. An example of an AXML document is given in Figure 1. An important feature is that the call of a web service may return data containing new embedded calls to further web services (see Figure 1). Each web service is specified using an *active*



AXML document   **Fig. 1.**   Same document after service calls

extension of WSDL [17], which defines its input and output type by means of AXML-schemes which in turn are an immediate extension of XML-schemes with additional tags for service calls. For instance, the specification of the service *www.meteo.fr* can be STRING → STRING while the specification of *www.aden.fr*

* Work supported by the DAAD and Egide under PROCOPE grant D/0205766.

can be $\emptyset \to$ OPERAS*MOVIES*OUTDOOR* where OPERA, MOVIES, OUTDOOR is either STRING* or a pointer to a web service.

Whenever a user or another application requests some data, the system must decide which data has to be materialized, in order to satisfy the request specification. An important issue is then which services are called and in which order. Assume for instance for our example in Figure 1 that there is a fee for each service call. If the request requires to minimize the overall costs, the system should first call *www.aden.fr* in order to get the list of events and only call the weather forecast if there is some available outdoor event. The requests we are considering in this paper ask for all available data of a given type as specified by an AXML-schema.

The system has access to local data, service specifications and a request specification. This can be modeled as follows (see [12, 1, 15]): (i) the local data is an AXML document corresponding to a labeled, unranked tree, (ii) the input/output type of a service specification is specified by a regular tree language, and, (iii) the requested data is also modeled by a regular tree language.

As this problem turns out to be computationally difficult, we consider a simpler version. Actually, even this simpler variant is undecidable without any further restrictions. First we assume that services do not have any input. Note that services with a fixed number of different inputs can be modeled by considering several different services, one per input option. Secondly, we assume that the output type consists of finitely many options, that is the regular language is in fact a disjunction of finitely many cases. Finally, we deal with strings instead of trees. This simplifies the combinatorics and allows a better understanding of the problem. Thus, the problem we consider here is stated as follows: given (i) a string, (ii) a set of service specifications of the form $A \to u_1 \mid \cdots \mid u_n$, where A is a letter and the u_i are strings, and (iii) a regular string language, can we decide which services to call and in which order, such that the string eventually obtained belongs to the regular language representing the target? We formalize this problem in terms of games. We discuss extensions of our framework to trees and full regular languages in the last section of the paper.

An *Active Context-Free Game* (CF-game) is played by two players (ROMEO and JULIET) on strings over a finite alphabet. Its rules are defined by a context-free grammar (CFG) and its target by a regular language given by a regular expression (equivalently, a non-deterministic automaton, NFA). In each move, JULIET jumps to a position of the current word and ROMEO rewrites the corresponding letter according to some rule of the grammar. JULIET wins a play if the string obtained belongs to the target language. The intended meaning is obvious: JULIET is the system, ROMEO the environment, the CFG corresponds to the service specification and the target language to the request specification.

We consider the complexity of deciding the existence of a winning strategy for JULIET in two variants. The first one, called *combined complexity*, means that both the specification of the game and the initial string are given as input. In the second variant, called *data complexity*, we fix a game specification and a target language, and the input consists of a string, only. It shows how the complexity behaves relatively to the length of the string. This can be motivated by the fact that the specification of the system is often fixed once and for all, while the data may frequently change. The data complexity measures then the difficulty of the problem after preprocessing the specification.

We show that without any restrictions, there is a fixed CF-game for which data complexity is already undecidable. Thus we consider simpler variants of the problem by restricting the set of rules, the regular target language, or the strategy. The above example already suggests two restrictions. First, both service calls give rise to one new service call tag, only. This means that the underlying CFG is linear. We also consider the more restricted case of unary grammars, where a service call may only return another service call or some data without any service call tag. A more realistic restriction, that is satisfied by the above example and probably by most applications, is that the iterated answer of service calls does not give back a tag with the same service call. This restriction corresponds to non-recursive CFGs and to non-recursive CFGs of given depth (bounded CFGs). The problem is decidable for all these restrictions, although it is intractable in some cases (e.g., ExpSpace for non-recursive grammars without uniform depth bound). We also consider left-to-right strategies where Juliet has to traverse the string from left to right. In the above scenario this amounts to having a heuristics for parsing the data tree only once, such that if the system decides not to call a service, it never comes back to this service again. This limits drastically the possibilities of the system but also decreases significantly the complexity of the problem. Combined with general CFGs the decision complexity is 2ExpTime and combined with non-recursive rules it is ExpTime. But for all other restrictions the complexity is at most PSpace. This restriction allows for a uniform decision procedure (and very efficient preprocessing as well) as an automaton accepting all winning configurations (strings) can be computed from the CF-game independently from the input string. To further decrease the complexity we also consider games where the specification of the target language is given as a deterministic automaton (DFA). In the case of bounded CFGs, and left-to-right strategies we end up with a tractable PTime decision procedure. This case seems rather restrictive at first sight, but it is general enough to handle many practical cases and it has been implemented in AXML [11].

Figures 2 and 3 summarize our results. The numbers in brackets refer to the corresponding theorem or lemma, respectively. All complexities are tight.

Rules Restriction	Combined Complexity NFA/DFA	Data Complexity
general	undecidable (1)	undecidable (1)
non-recursive	ExpSpace (4)	PSpace (5)
bounded	PSpace (5)	PSpace (5)
linear	ExpTime (5,1)	ExpTime (1)
unary	ExpTime (5,1)	ExpTime (1)

Fig. 2. Unrestricted strategies

Related work. For left-to-right strategies there is a tight connection with games on pushdown graphs [16] (see Propositions 1 and 2), which explains the decidability for arbitrary CFGs. A question related to the game problem is that of verifying properties of infinite graphs defined by CF-games (model-checking). Similar questions have been asked, e.g., for automatic graphs [4], process rewrit-

Rules Restriction	Combined Complexity NFA	Combined Complexity DFA	Data Complexity
general	2EXPTIME (3)	EXPTIME (2)	regular (2)
non-recursive	EXPTIME (8)	PSPACE (9)	regular (2)
bounded	PSPACE (6)	PTIME (7)	regular (2)
linear	PSPACE(10)	PSPACE (11)	regular (2)
unary	PSPACE (10)	PTIME (12)	regular (2)

Fig. 3. Left-to-right strategies

ing graphs [10] and ground tree rewriting graphs [8]. For instance, [8] considers CTL-like versions of the reachability problem in ground tree rewriting graphs. Graphs generated by CFGs on strings can be seen as a special case of ground tree rewriting graphs and therefore the undecidability result obtained in [8] follows from our Theorem 1.

Overview. The paper is organized as follows. Section 2 gives formal definitions and fixes the notation. It also describes a couple of extensions of the basic game which are used in the lower bound proofs. The results on arbitrary CFGs are given in Section 3. Non-recursive and linear CFGs are considered respectively in Section 4 and Section 5. Due to lack of space several proofs are omitted.

2 Definitions

A CF-game is a tuple $G = \langle \Sigma, R, T \rangle$, where Σ is a finite alphabet, $R \subseteq \Sigma \times \Sigma^+$ a finite set of *rules* and T a regular *target language*. Note that the rewriting rules do not allow the empty string on the right-hand side. We call a symbol A of Σ a *non-terminal* if it occurs on the left-hand-side of some rule in R, otherwise a *terminal*.

A *play* of the game G is played by two players, JULIET and ROMEO, which play in *rounds*. In each round, first JULIET selects a position and then ROMEO chooses a rewriting rule associated to the letter of the chosen position.

A *configuration* C of the game is a tuple (w, i, c) where w is a string (*the current word*), $i \leq |w|$ is a number (*the current position*) and c is either pos or rule. A *position choice* in configuration (w, i, pos) consists of selecting a position $j \leq |w|$ resulting in (w, j, rule). A *rule choice* in configuration $(a_1 \cdots a_n, j, \mathsf{rule})$ consists of replacing a_j by a string u such that $a_j \to u$ is a rule of G. The result is $(a_1 \cdots a_{j-1} u a_{j+1} \cdots a_n, j, \mathsf{pos})$. A play starts in an *initial configuration* $C_0 = (w, 1, \mathit{pos})$, for some string w.

The play stops and JULIET wins if after some round the resulting string is in T. Otherwise it goes on. ROMEO wins immediately, if JULIET chooses a position j, whose corresponding symbol is terminal. As usual, we say that JULIET has a *winning strategy* in configuration (w, i, c) if, no matter how ROMEO plays, T is reached within a finite number of moves.

Note that the winning condition for JULIET is in the first level of the Borel hierarchy (reachability of a set). By Martin's determinacy theorem, CF-games are thus determined i.e., from each configuration one of the two players must have a winning strategy, [6].

We consider the decision problem for JULIET to have a winning strategy in G on a string w. This comes in two flavors, *combined decision problem* and *data decision problem*. The *combined decision problem* is:

[Combined] INPUT: A CF-game $G = \langle \Sigma, R, T \rangle$, a string w
OUTPUT: True iff JULIET has a winning strategy in G on w.

The *data decision problem* associated with a CF-game G is:

[Data(G)] INPUT: A string w
OUTPUT: True iff JULIET has a winning strategy in G on w.

We say that JULIET has a *left-to-right winning strategy* if she can always choose a position which is bigger or equal to the position chosen in the preceding move. We call the set R of rules of a game *unary* if each rule is of the form $A \to B$ with $B \in \Sigma$. We call it *linear* if each right-hand-side of a rule contains at most one non-terminal. The set R is called *non-recursive* if no symbol A can be derived from A by a non-empty sequence. For a non-recursive set R we call the maximal depth d of a leaf in a derivation tree of R the *depth* of R. A CF-game G is *unary* (resp. *linear, non-recursive*) if its set R of rules is.

Extended games. In the lower bound proofs we make use of several extensions of the basic CF-game in order to simplify reductions. It turns out that the complexity of the decision problems does not change in many cases (we omit the proof of this result in this abstract). These extensions are

- *navigation constraints*: basically regular expressions associated with a rule, which restrict the possible position choices for JULIET in the next move. As an example, JULIET can be forced to choose the next position immediately to the right of the current one;
- *symmetric rule choice*: symbols for which JULIET, instead of ROMEO, chooses the rule;
- *concatenation of games*: games may consist of several successive phases.

For unrestricted strategies, every game G with all these features can be simulated by a usual game G' in polynomial time and every string w can be translated into a string w' such that JULIET wins (G, w) iff she wins (G', w'). Furthermore, unarity, linearity, non-recursiveness and even boundedness are preserved (but not all combinations, e.g. unarity and boundedness are not preserved at the same time). A similar result holds for left-to-right strategies. Navigation constraints behave in an analogous way. For symmetric rule choice, unarity is only preserved if all rules have navigation constraints. Concatenation of games does not seem to work here.

3 Unrestricted Rules

The section is divided into two parts. In the first one we consider unrestricted strategies, while the second one is devoted to left-to-right strategies.

Unrestricted strategies. We prove first that in general, both decision problems are undecidable. We will make use of the following lemma which establishes a close connection between computations and CF-games.

Lemma 1. *Let M be an alternating Turing machine with space bound $s(n)$ and initial state q_0. We can construct in polynomial time a unary game $G = \langle \Sigma, R, T\rangle$ such that the following assertion holds:*

For every input $w = a_1 \cdots a_n$ to M, Juliet *has a winning strategy for G on the string $w' = \$(q_0, a_1)a_2 \cdots a_n \sqcup^{s(n)-n} \#$ if and only if $w \in \mathcal{L}(M)$.*

The proof idea for the lemma above is to simulate a computation path of the alternating TM by letting Juliet play in existential configurations (symmetric rule choice) and Romeo in universal configurations. One single transition is simulated by a sequence of game moves, in which we use navigation constraints for forcing the players to rewrite the symbols affected by the transition.

From the theorem below it follows that both decision problems of CF-games are undecidable:

Theorem 1. *There exists a CF-game G for which the data decision problem is undecidable.*

Proof (Sketch): We reduce the *Post correspondence problem (PCP)* to Data(G), for some fixed game G. An instance of PCP is given by two sequences $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$ of finite words over $\{a, b\}^*$ for some $n \in \mathbb{N}$. The problem is to check whether there exist $m > 0$, $i_1, \ldots, i_m$, such that $u_{i_1} \cdots u_{i_m} = v_{i_1} \cdots v_{i_m}$.

The game is played on the string
$w = \$\, u_1 \& v_1 \& 01\, u_2 \& v_2 \& 001 \cdots u_n \& v_n \& 0^n 1 \& S \#$. We refer to the prefix of w before S by w_0. The string w encodes the PCP strings, together with their index. The symbol S will generate the solution $i_1, \ldots, i_m$.

The game has two phases. First, Juliet uses (non-linear, recursive) rules $\{S \to S_0 S_1 S, S_0 \to 0, S_1 \to 1, \# \to \sqcup\#\}$ to generate the string
$w_0 (0S_1)^{i_1} (S_0 1)^{i_2} \cdots (0S_1)^{i_m} S \sqcup^k \#$. In phase two it is checked that this string encodes a solution to PCP. As a deterministic TM can do this in linear space, Lemma 1 guarantees that it also can be done by a CF-game. □

Left-to-right strategies. In this section we consider only left-to-right strategies. Here, all problems become decidable, but the complexity depends on the representation of the target language. We first show that when the target language is given as a DFA, CF-games are closely related to pushdown games. We then show that when the target language is given as a NFA that there is an inherent exponential blowup.

A *reachability pushdown game* is played on a graph $G_{\mathcal{P}}$ associated with an alternating pushdown system $\mathcal{P} = \langle Q = Q_E \cup Q_A, \Gamma, \delta, F\rangle$. The nodes of $G_{\mathcal{P}}$ are the configurations $(q, u) \in Q \times \Gamma^*$ of $\mathcal{P}$. The set Q of states is partitioned into existential (Q_E, Eve's states) and universal (Q_A, Adam's states) states, and a node of $G_{\mathcal{P}}$ is existential (universal, resp.) if its control state is existential (universal, resp.). The transition relation $\delta \subseteq Q \times \Gamma \times Q \times \Gamma^*$ determines the edge relation $(q, u) \vdash (q', u')$ of $G_{\mathcal{P}}$.

Eve wins the reachability game if whatever Adam's choices are, she can reach a final configuration. It is known that deciding whether a configuration is winning is ExpTime-complete, [16]. Moreover, the set of winning configurations can be described by an alternating automaton of exponential size, [5, 14].

The next two propositions show the relation between pushdown games and CF-games with left-to-right strategies and DFA target language.

Proposition 1. *Given a game $G = \langle \Sigma, R, T \rangle$, where T is a DFA with initial state q_0, we can construct in polynomial time a pushdown system $\mathcal{P}$ such that* Juliet *wins the game G on w if and only if the configuration $c = (q_0, w\$)$ is winning in the reachability pushdown game.*

Proof: Let Q denote the set of states of the DFA T and δ_T its transition function. The states of $\mathcal{P}$ are $Q \cup \bar{Q} \cup \{f\}$, with Q existential states, $\bar{Q}$ universal ones and f the unique final state. The stack symbols are $\Sigma \cup \{\$\}$, where $\$ \notin \Sigma$.

For every pair $q \in Q$, $A \in \Sigma$ we have the transitions $\delta(q, A) = \{(\delta_T(q, A), \mathrm{pop}), (\bar{q}, A)\}$. The transitions correspond to Juliet either skipping the current position (pop), or selecting it and letting Romeo play next. For every pair $(\bar{q}, A)$ and every rule $A \to u$ of G we have a transition $(q, u) \in \delta(\bar{q}, A)$. These transitions correspond to Romeo choosing the corresponding rule in R. Finally, we add the transitions $(q, \$, f, \$)$ for every accepting state q of T. $\Box$

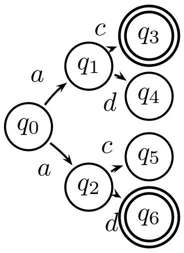
Proposition 2. *Given a pushdown system $\mathcal{P}$ we can construct in polynomial time a game $G = \langle \Sigma, R, T \rangle$, where T is a DFA, such that for any configuration $c = (q, A_1 \cdots A_n)$ of $\mathcal{P}$, c is winning in the pushdown game if and only if* Juliet *wins the game G on $w = (q, A_1)A_2 \cdots A_n$.*

Proposition 2 is shown by a game simulation using extended games. From propositions 1 and 2 and [16, 5, 14] we obtain immediately:

Theorem 2. *Given a game $\langle \Sigma, R, T \rangle$, where T is given as a DFA, it is* ExpTime*-complete to know whether* Juliet *has a winning left-to-right strategy.*

Moreover the set of input words for which Juliet *has a left-to-right winning strategy is regular and an alternating automaton that recognizes it can be constructed in exponential time from $\langle \Sigma, R, T \rangle$.*

Note 1. Proposition 1 above cannot be extended to CF-games with a non-deterministic target automaton. Indeed consider the following example.



In the game $G = \langle \Sigma, G, \mathcal{A} \rangle$ where $G = \{b \to c \mid d\}$, Juliet has a winning strategy on ab: rewrite b, as both ac and ad are accepting for $\mathcal{A}$. But in the pushdown system as constructed in the proof of Proposition 1, Adam has a winning strategy on $(q_0, ab\$)$: after reading a, Eve has to commit to state q_1 or to state q_2. Depending on Eve's choice at that state, Adam will choose respectively d and c for replacing b and thus will end in a non accepting configuration with state respectively q_4 and q_5.

Theorem 3. *Given a game $\langle \Sigma, R, T \rangle$ with T given by an NFA, it is* 2ExpTime*-complete to know whether* Juliet *has a winning left-to-right strategy.*

Proof (Sketch): As T can be transformed into an exponential size DFA, the 2ExpTime upper bound follows immediately from Proposition 1 and the fact that the winner in a pushdown game can be determined in exponential time in the size of the game. The lower bound is shown by simulating the behavior of an alternating exponential space Turing machine M on input x. Starting from $S\&x$, during a first phase the players keep rewriting the leftmost symbol only, generating a sequence of configurations of M. Each configuration is encoded by a sequence of (symbol,position)-pairs, where the position is an exponential size number encoded in binary. The alternation of M is mimicked by alternating between Juliet- and Romeo-choices (symmetric rule choice). In a second phase, it is checked in a single left-to-right pass, that the outcome of phase 1 really encodes an accepting computation of M on x. That is, Romeo gets the chance to object each position of the current string, i.e., replace by a special symbol. Then Juliet wins immediately if the objected position is correct, hence she wins the game if there is no error in the outcome of the first phase. It is crucial here that an NFA of polynomial size in n can express that $j \neq i$ and $j \neq i+1$, for two counter values $i, j < 2^n$. □

4 Non-recursive Rules

In this section we focus on non-recursive games. We also consider *bounded games*, i.e., with depth bounded by some constant d. Non-recursive games are important in practice because many applications do not have recursion in the calls of web services and the nesting of calls is small.

Unrestricted strategies. We first consider non-recursive sets of rules. We stress that the lower bound of the following theorem does not depend on whether the target language is coded as an NFA or as a DFA. Indeed we can show that in general, for unrestricted strategies, a CF-game with a target language given by an NFA can be reduced in polynomial time to a CF-game whose target language is given by a DFA, while preserving unarity, linearity and non-recursiveness.

Theorem 4. *It is* ExpSpace*-complete to decide whether* Juliet *has a winning strategy in a non-recursive game G on string w.*

Proof (Sketch): We show the upper bound by constructing an alternating TM deciding whether Juliet has a winning strategy in exponential time. It maintains on its tape the current game configuration. At each step it checks whether the string of the current configuration is in the target of the game. If yes it stops and accepts. If not it nondeterministically chooses a position to rewrite and universally branches over all possible rewritings. The time of each computation path is $O(m^d|w|)$, where m is the length of the maximal word occurring in G and d is the depth of G. The lower bound is obtained by simulating an exponential time alternating TM by a game. □

We now consider games with set of rules of depth bounded by some given d. The first lemma is used in lower bound proofs to get down from depth d to depth 1, the second lemma shows that the data decision problem is already hard for $d = 1$ and unary rules.

Lemma 2. *For each $d \geq 1$ there are polynomial-time computable functions $G \mapsto G'$ and $(w, G) \mapsto w'$ transforming any d-bounded CF-game G into a 1-bounded CF-game G', such that* Juliet *has a winning strategy in (G, w) if and only if she has a winning strategy in (G', w'). Furthermore, linearity, unarity and deterministic target are preserved.*

Lemma 3. *There is a unary CF-game $G = (\Sigma, R, T)$ of depth 1 such that Data(G) is* PSpace*-hard.*

Proof (Sketch): We use a reduction from the quantified Boolean satisfaction problem QBS. The input of QBS is a formula Φ in prenex normal form, with the quantifier-free part in 3CNF.

The extended game we construct consists of two phases and is played on a straightforward string encoding of Φ. The first phase is a left-to-right pass in which (i) each variable is rewritten by a truth value - by JULIET for existentially quantified variables and by ROMEO for universally quantified variables, (ii) a clause is selected by ROMEO, and, (iii) a literal in the clause by JULIET(symmetric rule choice). The second phase checks that the literal becomes true by the variable assignment. As R and T have to be independent of Φ, variable names are encoded as binary strings. Therefore, this check has to be done through the game by going back and forth between the value of each variable and the clause in which it is used. □

When d is fixed, $O(m^d|w|)$ is a polynomial bound. Therefore, from the above lemma and the upper bound proof of Theorem 4 we get:

Theorem 5. *For each d, given a CF-game G with rules of depth bounded by d and a string w it is* PSPACE*-complete to tell whether* JULIET *has a winning strategy for G on w. Furthermore, there is a game G of depth 1 for which Data(G) is* PSPACE*-complete.*

Left-to-right strategies. We continue with non-recursive rules of depth bounded by some d, but we now concentrate on left-to-right strategies. Recall Lemma 3 which shows that the data complexity of non-recursive games for unrestricted strategies is PSPACE-hard. With combined complexity PSPACE-hardness can now be obtained with only one pass.

Lemma 4. *For each $d \geq 1$, it is* PSPACE*-hard to tell, for a unary game $G = \langle \Sigma, R, T \rangle$ of depth d and a string w, whether* JULIET *has a left-to-right winning strategy.*

From Lemma 4 and an immediate adaptation of the proof of Theorem 5 for left-to-right strategies we obtain:

Theorem 6. *For each $d \geq 1$, it is* PSPACE*-complete to tell, for a game $G = \langle \Sigma, R, T \rangle$ of depth d, where T is given by an NFA, and a string w, whether* JULIET *has a winning left-to-right strategy for G on w.*

When the target language is given as a DFA the decision becomes tractable. The PTIME upper bound of the left-to-right, bounded, DFA target language case was already obtained in [11] using automata theoretical techniques. It is also the framework which has been implemented in AXML [11].

Theorem 7. *For each $d \geq 2$, given a game $G = \langle \Sigma, R, T \rangle$ of depth d and a string w where R is non-recursive and where T is a deterministic automaton, it is* PTIME*-complete to tell whether* JULIET *has a winning left-to-right strategy for G on w.*

Proof (Sketch): We prove the upper bound by constructing an alternating Turing machine deciding whether JULIET has a winning left-to-right strategy in logarithmic space. The machine has one pointer per level in the rewriting tree corresponding to a position of the input. Those d pointers are sufficient for the TM to specify the rightmost part of the current configuration which still needs to be processed.

The lower bound is proved by a reduction from the monotone Boolean circuit value problem [13]. Let C be a Boolean circuit. We can assume w.l.o.g. that all paths in C are alternating between **or** and **and** gates, have fan-in two and start/end with **and** gates [13]. From C we construct a DFA which accepts all strings over $\{l, r\}$ which describe paths from the output gate to an input gate which is 1. The extended game is played on a blank string of length depth(C) and Juliet and Romeo select such a path by rewriting each symbol by l or r. By doing so Juliet selects the predecessor of *or*-gates, Romeo of *and*-gates. The circuit is 1 iff Juliet can manage to end up in a 1-gate. Notice that the constructed game is also unary. □

In the case of non-recursive rules without a uniform depth bound the combined complexity is one level higher.

Theorem 8. *It is* ExpTime-*complete to know whether* Juliet *has a left-to-right strategy for G on w, for G non-recursive and target given by an NFA.*

Theorem 9. *It is* PSpace-*complete to know whether* Juliet *has a left-to-right strategy for G and w, if G are non-recursive and the target is given by a DFA.*

5 Linear Rules

In this section we focus on linear and unary games. Recall from our example that in practice, service calls often generate a single subsequent call, which motivates linear CFGs.

Unrestricted strategies. From Lemma 1 it follows immediately that the complexity of the data decision problem for *unary* games is ExpTime-hard. The following lemma shows that the combined decision problem for *linear* games can be done in ExpTime. Hence, for unary and linear games with unrestricted strategies all decision problems are ExpTime-complete.

Lemma 5. *Given a game $G = \langle \Sigma, R, T \rangle$ where R is linear, and a string w, one can tell in* ExpTime *whether* Juliet *has a winning strategy.*

Proof (Sketch): Let k be the number of non-terminal symbols occurring in w. By linearity of R this will be an upper bound on the number of non-terminal symbols during the game. We construct an alternating polynomial space Turing machine that decides whether Juliet has a winning strategy. Alternation is used to mimic the CF-game as usual and memory is used to store the current configuration. For the latter the machine needs only to maintain a sequence $f_1 A_1 f_2 A_2 \cdots A_k f_{k+1}$ where the A_i are the non-terminal letters of the current configuration while the f_i are transition relations of T corresponding to the words between successive non-terminal symbols. This requires space $O(k|Q|^2)$. □

Left-to-right strategies. We now consider left-to-right strategies for unary and linear CF-games.

Theorem 10. *It is* PSPACE*-complete to tell, given a unary (resp. linear) CF-game whether* JULIET *has a left-to-right-winning strategy.*

Proof (Sketch): For the lower bound the unary case suffices, for which it follows from Lemma 4. For the upper bound it is enough to consider the linear case. We check in NPSPACE whether ROMEO has a winning strategy. Hence, we guess the moves of ROMEO and, using backtracking, we cycle through all possible moves of JULIET. I.e., for the first symbol to replace, we first compute what happens if JULIET jumps after one move of ROMEO, then after the second, and so on. In order to do so, we only need to store a polynomial number of game configurations. Each configuration is of polynomial size as in Lemma 5. □

When the target language is given as a DFA the complexity decreases in the unary case, but not in the linear case:

Theorem 11. *It is* PSPACE*-complete to tell, given a linear CF-game with target language given by a DFA whether* JULIET *has a left-to-right-winning strategy.*

Proof (Sketch): The upper bound follows from Theorem 10. We prove the lower bound by simulating the behavior of a polynomial space Turing machine M by a CF-game with linear rules and a deterministic target automaton. Using linear recursive rules of the form $S \to Sa$ ROMEO produces a sequence of configurations of M. When he does a mistake JULIET immediately stops. Therefore the mismatch comes from the previous configuration which is within polynomial distance (recall that M uses only polynomial space) from the beginning of the current string. The target language, a polynomial size DFA, can therefore check the mistake by proper counting and comparing the corresponding positions. □

Theorem 12. *It is* PTIME*-complete to tell, given a unary CF-game with target language given by a DFA whether* JULIET *has a left-to-right-winning strategy.*

Proof (Sketch): The lower bound is done as in the proof of Theorem 7. For the upper bound we construct an alternating Turing machine deciding whether JULIET has a winning strategy in logarithmic space. Because the grammar is unary the length of the word never increases and therefore the input word can be processed letter by letter while reading it. For each letter the TM maintains on its tape a pointer to the current state in the target automaton and a pointer to the current candidate letter for replacing the current position. Looping over the alphabet is avoided by using a counter of logarithmic size. Alternation is used in a standard way to mimic the CF-game. □

6 Discussion

We have seen that in general is it undecidable to tell who wins a CF-game. We have also seen several restrictions on the set of rules and on the strategy which imply decidability. A natural interesting situation not considered in this paper is the case where the target language T is finite. This is often the case in our scenario, as a user may require all data looking exactly like this or that, with no other options. If no ϵ-rules are allowed, the game is obviously decidable in

ExpTime (APSpace) as no useful configuration can be larger than the size of T. It is open whether this bound is tight. If ϵ-rules are allowed it is not even clear whether the game is decidable.

As mentioned in the introduction the initial motivation of this work was to consider trees and rules defined using *extended* context-free grammars (rules of the form $a \to R_a$ where R_a is a regular language). For trees each rule would rewrite a leaf labeled a into a finite tree (into a regular language in the extended case). We can show that all the results presented in this paper extend to trees. However the situation is more complex for extended CFG rules. We can extend all our results with left-to-right strategies to these grammars but for unrestricted strategies even the non-recursive case can be shown to be undecidable.

Knowing that there exists a winning strategy is one thing. In practice the system needs to know which web service it should call and in which order. This correspond to extracting a winning strategy of a CF-game when it exists. We can show that this is always possible within the same complexity bounds as for the decision problem.

Acknowledgment. We are grateful to Tova Milo who brought the problem to our attention and to Tova Milo and Omar Benjelloun for the time they spent explaining us the beauty of AXML.

References

1. S. Abiteboul. *Semistructured Data: from Practice to Theory.* In *LICS'01*, IEEE Comp. Soc. 2001.
2. S. Abiteboul, A. Bonifati, G. Cobena, I. Manolescu, and T. Milo. *Dynamic XML documents with distribution and replication.* In *SIGMOD'03*, pages 527-538, ACM 2003.
3. Active XML. *http://www-rocq.inria.fr/verso/Gemo/Projects/axml.*
4. A. Blumensath and E. Grädel. Automatic structures. In *LICS'00*, pages 51–62, IEEE Comp. Soc. 2000.
5. Th. Cachat. *Symbolic Strategy Synthesis for Games on Pushdown Graphs.* In *ICALP'02*, LNCS 2380, pages 704-715, Springer, 2002.
6. E. Grädel, W. Thomas, and Th. Wilke, eds. *Automata, Logics, and Infinite Games.* Springer, 2002.
7. Jelly: Executable XML. *http://jakarta.apache.org/commons/sandbox/jelly.*
8. Ch. Löding. *Infinite graphs generated by tree rewriting.* PhD thesis, RWTH Aachen, 2003.
9. Macromedia Coldfusion MX. *http://www.macromedia.com/.*
10. R. Mayr. *Process rewrite systems.* In *Theoretical computer science* 156(1-2):264-286, 2000.
11. T. Milo, S. Abiteboul, B. Amann, O. Benjelloun, F. Dang Ngoc. *Exchanging Intensional XML Data.* In *SIGMOD'03*, pages 289-300, ACM 2003.
12. F. Neven. *Automata, Logic, and XML.* In *Proc. of CSL'02*, LNCS 2471, pages 2-26, Springer, 2002.
13. C. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.
14. O. Serre. *Note on winning positions on pushdown games with ω-regular conditions.* In *Information Processing Letters* 85:285-291, 2003.
15. V. Vianu. *A Web Odyssey: From Codd to XML.* In *PODS'01*, ACM 2001.
16. I. Walukiewicz. *Pushdown Processes: Games and Model-Checking.* In *Information and Computation* 164(2), 2001, pages 234-263.
17. Web services. *http://www.w3.org/2002/ws.*