Active Context-Free Games

Anca Muscholl, Thomas Schwentick, Luc Segoufin

Active Context-Free Games*

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

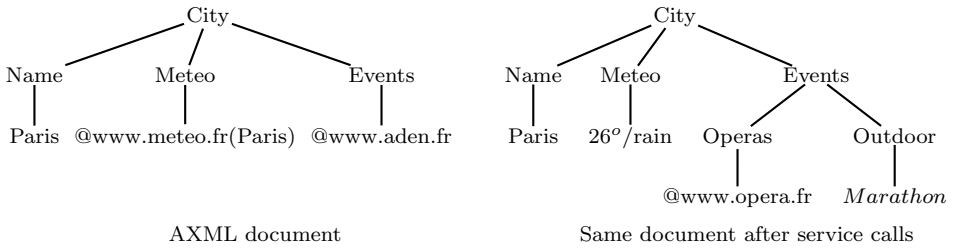
[1] LIAFA, Université Paris VII, 2 pl. Jussieu, F-75251 Paris
[2] Philipps-Universität Marburg, FB Mathematik und Informatik, D-35032 Marburg
[3] INRIA, Parc Club Orsay Univ., ZAC des vignes, 4 rue J. Monod, F-91893 Orsay

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. The complexity of deciding the existence of winning strategies for JULIET is investigated, 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* extension of WSDL [17],



AXML document — Same document after service calls

Fig. 1.

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* can be $\emptyset \rightarrow$ 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

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

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. The main simplification is that we work on strings instead of trees. This allows a better understanding of the complexity of our problem. Thus, the questions considered here are 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 is guaranteed to belong 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 Section 7 and in the last section.

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 it is undecidable to know if Juliet has a winning strategy. This leads us to consider simpler variants of the problem by restricting the set of rules, the regular target language, and/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. Another 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 or even to non-recursive CFGs of fixed 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. The left-to-right 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 of 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 above NC^1 are tight.

Rules Restriction	Combined Complexity NFA/DFA	Data Complexity
general	undecidable (4.2)	undecidable (4.2)
non-recursive	ExpSpace (5.1)	PSpace (5.3)
bounded	PSpace (5.3)	PSpace (5.3)
linear	ExpTime (6.1,6.2)	ExpTime (6.1,6.2)
unary	ExpTime (6.1,6.2)	ExpTime (6.1,6.2)

Fig. 2. Unrestricted strategies

Rules Restriction	Combined Complexity NFA	Combined Complexity DFA	Data Complexity
general	2ExpTime (4.3)	ExpTime (4.6)	NC^1 (4.6)
non-recursive	ExpTime (5.6)	PSpace (5.9)	NC^1 (4.6)
bounded	PSpace (5.5)	PTime (5.8)	NC^1 (4.6)
linear	PSpace(6.4)	PSpace (6.5)	NC^1 (4.6)
unary	PSpace (6.4)	PTime (6.6)	NC^1 (4.6)

Fig. 3. Left-to-right strategies

We then discuss the case where the output set of choice is no longer finite, but given as a regular set. In that case, the set of rules is given as an *extended* context-free grammar (ECFG) where each rule has a regular expression as right-hand side. We show that in this case unrestricted winning strategies are already undecidable even for non-recursive rules of depth two. However for left-to-right strategies the decision problem has the same complexity as in the finite case.

Related work. For left-to-right strategies there is a tight connection with games on pushdown graphs [16] (see Propositions 4.4 and 4.5), 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 rewriting graphs [10] and ground tree rewriting graphs [8]. For instance, [8] considers CTL and game-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.

Overview. The paper is organized as follows. Sections 2 and 3 give formal definitions and fix the notation. We also describe a couple of extensions of the basic game which are used in the lower bound proofs. In order to improve the readability of the paper the proofs of these lemmas are postponed until Section 8. The results on arbitrary CFGs are given in Section 4. Non-recursive and linear CFGs are considered respectively in Section 5 and Section 6. Section 7 is devoted to extended context-free rules.

2 Basic definitions and notations

2.1 Context-free games

A *context-free game* (CF-game, for short) 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 is a (non-deterministic) finite automaton representing 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, who play in *rounds*. A play consists of applying rewriting steps to a given string over Σ. In each round, first Juliet selects a position in the string and then Romeo chooses a rewriting rule associated with the letter of the chosen position.

A *configuration* C of the game is a pair (w, i) where w is a string (*the current word*) and $i \leq |w|$ is a number (*the current position*). A *position choice* in configuration $(a_1 \cdots a_n, i)$ consists of selecting a position $j \leq n$, a *rule choice* consists of replacing a_j by a string u such that $a_j \to u$ is a rule of G. The resulting configuration is $(a_1 \cdots a_{j-1} u a_{j+1} \cdots a_n, j)$. For simplicity, we often write for short (u, v) for a configuration $(uv, |u|+1)$. Similarly, if i is clear from the context, we often just write w for a configuration (w, i).

A play on string w starts in the *initial configuration* $C_0 = (w, 1)$. The play stops and Juliet wins if after some round the resulting string is in the language $L(T)$ accepted by 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) if, no matter how Romeo plays, a string in $L(T)$ is reached within a finite number of moves. To express that Juliet has a winning strategy in the game G on w we simply say Juliet *wins* (G, w).

If the target automaton is a deterministic finite automaton (DFA), we call a CF-game *deterministic*. For non-deterministic games we often specify the target language by a regular expression instead of a NFA. It is well-known that regular expressions can be transformed into equivalent NFAs of linear size.

From the point of view of two-player games on (infinite) graphs the CF-games considered here have a very simple winning condition, which belongs to a low level of the Borel hierarchy

(reachability of a set). By Martin's determinacy theorem [6], CF-games are thus determined i.e., from each configuration one of the two players must have a winning strategy.

We consider the decision problem for JULIET having 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 in* (G, w) if she wins with a strategy in which the sequence of chosen positions is non-decreasing. I.e., if (v, j) is a configuration reached from (u, i) then $j \geq i$.

A rule is *linear* if its right-hand side contains at most one non-terminal. A rule is *unary* if it is of the form $A \to B$ with $B \in \Sigma$ or $A \to w$ and w only contains terminal symbols. A set of rules is linear (resp., unary) if each rule is linear (resp., unary). The *width* of a rule is the number of symbols on its right-hand side.

A set of rules is called *non-recursive* if no symbol can be derived from itself by a non-empty sequence of rewritings. 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. If the depth of R is at most d, then we call it d-bounded. The width of R is the maximum width of a rule in R.

A CF-game is *unary* (resp. *linear, non-recursive, d-bounded*) if its set of rules is.

2.2 Complexity

Throughout the paper we use standard complexity classes, such as deterministic polynomial time (PTIME), polynomial space (PSPACE), deterministic exponential time (EXPTIME), exponential space (EXPSPACE) and deterministic double exponential time (2EXPTIME). We also use the fact that regular languages belong to the class NC^1.

Due to the alternating nature of games, our proof techniques are often based on alternating Turing machines (ATM), which we denote usually as $M = \langle\, Q = Q_e \cup Q_a, \Gamma_0, \Gamma, q_0, \delta, F \,\rangle$ with set of states Q partitioned into existential states Q_e and universal states Q_a, input alphabet Γ_0, tape alphabet $\Gamma \supseteq \Gamma_0$, transition relation $\delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{\mathrm{Left}, \mathrm{Right}\}$, initial state q_0 and set of final states F. The acceptance of such a device is defined as usual through accepting run trees (i.e., trees labeled by configurations, such that existential configurations have one child labeled by some successor configuration, and universal ones have all their successor configurations as children).

We frequently represent configurations of a Turing machine M by strings over the alphabet $\Gamma \cup (Q \times \Gamma)$ with exactly one symbol $(q, A) \in Q \times \Gamma$, representing the state of M and the symbol under the head. For a transition d of M we write $ABC \to_d D$, with $A, B, C, D \in \Gamma \cup (\Gamma \times Q)$, if the application of d in a configuration $uABCv$ leads to a configuration string, in which B is replaced by D. For example, if $d = (q_1, A, q_2, B, \mathrm{Right})$ then $(q_1, A)CC \to_d (q_2, C)$ and $C(q_1, A)C \to_d B$. We write $ABC \to_M D$ if $ABC \to_d D$, for some $d \in \delta$.

We use furthermore standard relations between alternating and non-alternating complexity classes: PSPACE (EXPSPACE, resp.) equals alternating polynomial (exponential, resp.) time and EXPTIME (2EXPTIME, resp.) equals alternating polynomial (exponential, resp.) space. Also recall that on the level of finite automata, alternation does not increase the power.

To improve readability, we usually put symbols from a cartesian product into a framed box. For instance, a symbol (q_1, A) would often be displayed as $\boxed{q_1, A}$.

3 Extended games and normal forms

The lower bound proofs of the following sections we reduce various problems to the combined decision problem and the data decision problem for games with restricted rules and/or left-to-right strategies. The games constructed there make use of a couple of encoding techniques that force one of the players to play in a certain way. In order to reduce the technical complication of these reductions we isolate some of these techniques in the current section by means of extending the original CF-games in several ways. Furthermore, we exhibit two normal forms for CF-games. The proofs of these results are given in Section 8.

More specifically, we consider the following three extensions of the basic game.

- Navigation constraints that restrict the possible position choices of JULIET;
- Symmetric rule choice, i.e., non-terminals for which JULIET (instead of ROMEO) is allowed to choose a rule;
- Games that consist of several phases.

The normal form results are as follows.

- For unrestricted strategies, a game with target NFA can be transformed in polynomial time into a (basically) equivalent game with target DFA.
- For each fixed d, games of depth d can be transformed into (basically) equivalent games of depth 1.

The most frequent proof technique is to let a player *object* whenever the other player does not play as required. For instance, ROMEO can prevent JULIET from choosing certain positions at certain times in a game. For this purpose, non-terminals A have a rule $A \to A^\perp$ which can be used by ROMEO to indicate a forbidden position choice of JULIET (symbols of the form $A^\perp$ are called *objection symbols*). The target automaton T of the game ensures that the objection symbols are used only under specific circumstances.

3.1 Extended games with unrestricted strategies

A *navigation restriction* is a triple $\rho = (d, r, b)$ consisting of

- a *direction* $d \in \{\leftarrowtail, \rightarrowtail\}$,
- a regular expression r over Σ, and
- a symbol $b \in \Sigma \cup \{*\}$.

In a *CF-game* $G = \langle \Sigma, R, T, N \rangle$ *with navigation restrictions* each rule α is associated with a set $N(\alpha)$ of navigation restrictions. Furthermore, $N(\epsilon)$ is a set of navigation restrictions which apply to the first position choice of a play.

Assume ROMEO selects a rule $A \to u$ with restriction set $\{\rho_1, \ldots, \rho_m\}$ in configuration (v, Aw) resulting in the configuration (v, uw). The next position choice of JULIET must respect one of the navigation constraints ρ_i. It respects the constraint $\rho_i = (\rightarrowtail, r, b)$ if JULIET chooses a position j such that the string between u and the chosen position, i.e., the prefix of w of length $j - |uv| - 1$ is in the regular language $L(r)$ of the expression r, and the symbol a at

position j *matches* b, i.e., either $b = *$ or $a = b$. Analogously, it respects a constraint $(\leftharpoondown, r, b)$ if the suffix of v of length $|v| - j$ is in $L(r)$ and the chosen symbol matches b.

It should be noted that, in particular, the new position has to be outside the right-hand side u of the rule. Note also that $r = \epsilon$ means that JULIET must choose the next position immediately to the right or to the left, respectively. Finally note that JULIET looses immediately if she cannot find a next position satisfying the navigation constraints.

If $N(\alpha)$ is empty for a rule α, then the choice of JULIET is not restricted.

For a constraint $(\rightharpoonup, r, b)$ in $N(\epsilon)$ the situation is similar: the length $(j-1)$-prefix of the string has to be in $L(r)$ and the j-th symbol must match b.

It should be also noted that in a CF-game $G = \langle \Sigma, R, T, N \rangle$ with navigation restrictions, the same rule of R might exist several times with different associated navigation restrictions. For a deterministic game, the navigation rules have DFA instead of regular expressions.

The second extension allows non-terminals for which JULIET is allowed to select a rule. A *CF-game with symmetric rule choice* is a CF-game in which a set Σ_E of non-terminals is distinguished. Here, the subscript E indicates that these non-terminals belong to JULIET. Whenever JULIET selects a position with a non-terminal A from Σ_E, she is the one to choose a rule $A \to u$ in the next step.

The last extension allows games to consist of several (but constantly many) phases. Each phase i has its own set R_i of rules and its own goal T_i. Furthermore, each phase i has an additional regular set W_i which limits the possibility to go to the next phase. Whenever a configuration (w, j) with $w \in W_i$ is reached in phase i, JULIET can choose to go to the next phase of the game. Formally, an *extended CF-game* G consists of

- an alphabet Σ,
- a number k of phases,
- k tuples $(R_i, T_i, N_i, \Sigma_{E,i}, W_i)$, such that for each $i \leq k$, R_i is a set of rules, T_i is a regular set of winning strings, for each rule α, $N_i(\alpha)$ is a set of navigation constraints, $N_i(\epsilon)$ is the set of navigation constraints for the first move in phase i, $\Sigma_{E,i}$ is a set of non-terminals which allow JULIET to choose a rule, and W_i is a regular set of strings that allow JULIET to enter the next phase.

JULIET wins if for *some* i, in phase i a string in T_i is reached. Note that the set W_k has no meaning, but we keep it for simplicity.

The following lemma shows that the three extensions presented above do not enlarge the expressive power of games. Note that the (pure) CF-game constructed in the lemma only depends on the extended game G, i.e., not on the initial string v. This is required for all data complexity reductions.

Lemma 3.1. *There are polynomial-time computable functions mapping extended CF-games G to CF-games G' and pairs (G, w) to strings w' such that* JULIET *wins (G, w) if and only if* JULIET *wins (G', w'). If G unary, linear or non-recursive then G' can be guaranteed to have the same property. If a non-recursive G is d-bounded then G' can be chosen $O(d)$-bounded.*

It should be stressed that Lemma 3.1 does not preserve linear and non-recursive rule sets at the same time.

3.2 Extended games with left-to-right strategies

A similar result holds in the context of left-to-right strategies. Nevertheless, the proof techniques are different (actually simpler) but not all rule types can be preserved. In the context of

left-to-right strategies, we do not consider games with several phases since these strategies are essentially one-phase. It will turn out that for left-to-right strategies the coding of the target language does matter. For instance we will see that, without restrictions on the rules, if the target language is given by an NFA the combined complexity problem is 2ExpTime-complete while it is in ExpTime when the target language is deterministic.

Lemma 3.2. *For each CF-game $G = \langle \Sigma, R, T, N, \Sigma_E \rangle$ with symmetric rule choice and navigation constraints a CF-game $G' = \langle \Sigma', R', T' \rangle$ can be constructed in polynomial time such that, for each string w,* Juliet *wins (G, w) with a left-to-right strategy if and only if* Juliet *wins (G', w) with a left-to-right strategy. Furthermore, the following statements hold.*

(a) If, for some d, G is $O(d)$-bounded (non-recursive, resp.) then G' is $2d$-bounded (non-recursive, resp.).
(b) If G is unary (linear, resp.) and all Σ_E-rules have a navigation restriction, then G' is also unary (linear, resp.).
(c) If T is a DFA and each rule of R has at most one navigation restriction then T' is a DFA.

3.3 Normal forms for games

The next lemma shows that for unrestricted strategies, the coding of the target language does not affect the complexity. Indeed it shows that it is possible to reduce any CF-game with an NFA target to a CF-game with a DFA target.

Lemma 3.3. *For each CF-game $G = \langle \Sigma, R, T \rangle$ a CF-game $G' = \langle \Sigma', R', T' \rangle$ can be constructed in polynomial time such that T' is given by a DFA and for each string w,* Juliet *wins (G, w) if and only if* Juliet *wins $(G', w\$)$, where $\$$ is an additional symbol not occurring in Σ. Furthermore, if G is unary, linear or non-recursive then G' can be guaranteed to have the same property. If G is d-bounded then G' can be chosen $O(d)$-bounded.*

We finally state the following lemma which enables us in lower bound proofs to reduce the depth of the constructed game to 1.

Lemma 3.4. *For each $d \geq 1$ there are polynomial-time computable functions transforming d-bounded CF-games G into 1-bounded CF-games G', and pairs (G, w) into strings w' such that* Juliet *wins (G, w) if and only if* Juliet *wins (G', w'). The same statement holds with respect to left-to-right strategies. Furthermore G' can be enforced to be unary or linear if G has the corresponding property and it is deterministic in case G is deterministic.*

4 Unrestricted rules

In this section, we consider games in which the rules are not restricted. The section is divided into three parts.

In Subsection 4.1, we consider unrestricted strategies and obtain undecidability even in the data complexity case. The basic idea to show this result is as follows. To each ATM M we can associate a CF-game G such that M accepts a string w in space m iff Juliet wins $(G, \$w \sqcup^{m-|w|} \#)$ (Lemma 4.1). Undecidability is obtained by choosing an ATM M with an undecidable language $L(M)$ and adding an initial phase to G, in which sufficiently many blanks $\sqcup$ are added (sufficient for the run of M on w, if M accepts w).

The restriction to left-to-right strategies however results in decidable decision problems. In Subsection 4.2 it is shown that the combined complexity in this case is 2ExpTime-complete.

Finally, in Subsection 4.3 we consider left-to-right strategies for deterministic games. The specification of the target language by a DFA results in a considerable improvement in complexity, ExpTime-complete. In fact, such games are strongly related to pushdown games which have the same combined complexity. This relationship further implies that for left-to-right strategies the sets Data(G) are regular, therefore the data complexity for games with left-to-right strategies is in NC^1.

4.1 Unrestricted rules and unrestricted strategies

We prove first that in general, both the combined and the data decision problem are undecidable. The proof uses the following lemma which establishes a close connection between alternating computations and CF-games. The fact that the game constructed in this lemma is even unary will be used later for the proof of Corollary 6.1.

Lemma 4.1. *Let M be an alternating Turing machine with space bound $s(n)$ and initial state q_0. One can construct in polynomial time a unary game $G = \langle \Sigma, R, T \rangle$ such that, for every input $w = a_1 \cdots a_n$ to M it holds:*

M accepts w if and only if Juliet *wins the game G on $\$\boxed{q_0, a_1} a_2 \cdots a_n \sqcup^{s(n)-n} \#$.*

Proof. The proof idea is to simulate a computation path of the ATM 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.

Let $M = \langle\, Q = Q_e \cup Q_a, \Gamma_0, \Gamma, q_0, \delta, F \,\rangle$ and $w \in \Gamma_0^*$. Recall that Q_e (Q_a, resp.) denotes the set of existential (universal, resp.) states, and $\delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{\mathrm{Left}, \mathrm{Right}\}$ is the transition relation of M. Let also $\sqcup$ denote the blank symbol of M. Without restriction, we assume that each state r of M can be entered either only from the left, or only from the right. That is, if $(q, A, r, B, \mathrm{Right}) \in \delta$, then $(q', A', r, B', \mathrm{Left}) \notin \delta$ for all q', A', B' (symmetrically for left moves).

The alphabet Σ of the game is defined as

$$\Sigma = \Gamma \cup \{\$, \#\} \cup \{\boxed{q, A}, \boxed{q, A, \leftarrow}, \boxed{q, A, \rightarrow} \mid q \in Q, A \in \Gamma\}.$$

We explain first the goal of the game informally. Suppose that the current string is of the form $(\$u, \boxed{q, A} C\, v\#)$, corresponding to a configuration $u(q, A)Cv$ of M with tape content $uACv$, state q and the head on A ($u, v \in \Gamma^*$, $A, C \in \Gamma$). The execution of the transition $(q, A, r, B, \mathrm{Right})$ (i.e., "replace A by B and go to the right into state r") should correspond to the following sequence of game configurations.

$$(\$u, \boxed{q, A} Cv\#) \to (\$u \boxed{r, B, \rightarrow}, Cv\#) \to (\$u, \boxed{r, B, \rightarrow} \boxed{r, C} v\#) \to (\$uB, \boxed{r, C} v\#)$$

This can be accomplished by the following rules.

- $\boxed{q, A} \to \boxed{r, B, \rightarrow}$ with constraint $(\rightarrowtail, \epsilon, *)$ (go to the right),
- $C \to \boxed{r, C}$ with constraint $(\leftarrowtail, \epsilon, *)$ (go to the left),
- $\boxed{r, B, \rightarrow} \to B$ with constraint $(\rightarrowtail, \epsilon, *)$.

We used the fact that each state can be entered from a unique direction in order to associate a unique navigation constraint with the rule $C \to \boxed{r, C}$. Also note that each step of the ATM must be simulated through consecutive game moves (otherwise JULIET looses). The case of left moves is similar.

The existential states $q \in Q_e$ correspond to symbols $\boxed{q, A} \in \Sigma_E$ for which JULIET chooses the rule for rewriting. Correspondingly, universal states $q \in Q_a$ correspond to symbols $\boxed{q, A} \notin \Sigma_E$ for which ROMEO chooses the rewriting rule.

It should be also noted that JULIET never has a choice of the position. It is always determined by the transition of M and either immediately to the right or to the left of the previous position.

Besides that, for existential states q she can select the next transition. ROMEO, on the other hand, can choose transitions for universal states and also rules of the form $C \to \boxed{r, C}$, as in the example above. To enforce that he always takes a rule with the correct state r in $\boxed{r, C}$, T accepts all strings of the forms $\$u\boxed{p, A, \to}\boxed{q, B}v\#$ and $\$u\boxed{q, B}\boxed{p, A, \leftarrow}v\#$ with $u, v \in \Gamma^*$ and $p \neq q$.

The constraints in $N(\epsilon)$ force JULIET to choose position 2 in the initial string. To model acceptance of M, the target T also covers all strings of the form $\$u\boxed{f, A}v\#$ with $u, v \in \Gamma^*$ and f an accepting state of M.

Finally, the statement of the lemma follows from Lemma 3.1. □

Using Lemma 4.1 it is now very easy to show that both decision problems of CF-games are undecidable.

Theorem 4.2. *There exists a CF-game G for which the data decision problem is undecidable. Thus, the combined decision problem for CF-games is undecidable as well.*

Proof. Let M be a deterministic Turing machine with an undecidable language $L(M)$, e.g., M might be chosen as a universal TM. We describe how M can be transformed into an extended game G.

For each string $w \in L(M)$ there is an $m \geq |w|$ such that w is accepted by M in at most m steps and with space m.

For an input $w = a_1 \cdots a_n$ to M, let $w' = \boxed{q_0, a_1}a_2 \cdots a_n$, where q_0 is the initial state of M. We consider the following extended game G on string $\$w'S\#$.

In the first phase of G, JULIET transforms $\$w'S\#$ into the string $\$w' \sqcup^{m-n} \#$. This is done by applying the rules $S \to S\sqcup\ m - n - 1$ and $S \to \sqcup$. The symbol S is in Σ_E, hence JULIET can apply the rule as often as she likes.

The rest of the game is as in the proof of Lemma 4.1. Therefore, JULIET wins the game on $\$w'S\#$ if and only if $w \in L(M)$.

Note that if $w \notin L(M)$ the computation of M might be infinite and the space it uses might be unbounded. This will result in a situation in which JULIET looses, as soon she cannot respect the navigation constraint "go to the right".

By application of Lemma 3.1 the game G can be transformed into a pure CF-game.

□

4.2 Unrestricted rules and left-to-right strategies

In this subsection we show that for left-to-right strategies the combined decision problem is decidable, although the complexity is high.

Theorem 4.3. *The combined decision problem for left-to-right strategies when the target language T is given by an NFA is* 2ExpTime*-complete.*

Proof. As the NFA for T can be transformed into an exponential size DFA, the 2ExpTime upper bound follows immediately from Theorem 4.6 below.

The lower bound is shown by simulating the behavior of an alternating exponential space Turing machine M on input w by an extended CF-game consisting of a construction stage and a testing stage.[4] During the construction stage, starting from $S\&w$, the players keep rewriting the leftmost symbol S only, generating a sequence of configurations of M (in reversed order). 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). Romeo chooses the universal transitions, while Juliet chooses the existential transitions and the configurations according to the selected transition.

In the test stage, it is checked in a single left-to-right pass, that the outcome of the construction stage encodes an accepting computation of M on w. That is, Romeo gets the chance to object at each position of the current string. If he does this Juliet wins if the objected position is correct. It will be crucial that an NFA of polynomial size in N can express that $j \neq i$ and $j \neq i+1$, for any counter values $0 \leq i, j < 2^N$, encoded as binary strings.

Now we describe the construction of G more formally. Let $M = \langle\, Q = Q_e \cup Q_a, \Gamma_0, \Gamma, q_0, \delta, F \,\rangle$ be the alternating Turing machine working in space $2^{p(n)}$ on inputs of size n, for some polynomial p. Let w be an input of size n and let $N = p(n)$. For simplicity we assume that M always alternates between existential and universal configurations, and that the initial configuration is existential.

A computation of M on w will be represented as a sequence of configurations, separated by an encoding of the transition leading from one configuration to the next one. For each configuration we attach after each symbol its position in binary. That is, the word generated after the first part must be of the form

$$v = UC_t \# d_{t-1} \$ C_{t-1} \# d_{t-2} \$ \cdots \# d_0 \$ C_0 \& w$$

where U is a symbol indicating that the construction stage is finished, each C_i is a string encoding a configuration of M, and each d_i is a transition of M. Such a word is *correct* if C_0 is the initial configuration of M on w, C_t is accepting and each C_{i+1} follows from C_i using the transition d_i. It should be noted that the latter condition implicitly guarantees that the configurations alternate between universal and existential ones.

Each configuration $C_i = A_0 \cdots A_{2^N-1} \in \Gamma^* \Gamma_Q \Gamma^*$ is encoded as

$$A_0 \mathrm{bin}(0) \cdots A_{2^N-1} \mathrm{bin}(2^N - 1)$$

where $\Gamma_Q = \{(A, q) \mid A \in \Gamma, q \in Q\}$ and where $\mathrm{bin}(j)$ is the length N binary encoding of $0 \leq j < 2^N$ with the *rightmost* bit being the least significant one.

The alphabet Σ of the game consists of

- the symbols $S, U, U', U_e, U_a, 0, 1, \bar{0}, \bar{1}, \#, \$, \&, \bot$,
- the set $Y = \Gamma \cup \Gamma_Q$,

[4] These stages should not be confused with the *phases* of extended games.

– a copy Δ of the transition relation δ,
– and some objection symbols which will be defined later.

The set Σ_E (for which JULIET chooses the rewriting rules) is $\{U_e, S\}$.

The rewriting rules for the construction stage contain:

$$\begin{aligned} S &\to SA \mid S0 \mid S1 \mid U_e\$ \mid U_a\$ \mid U \qquad \text{where } A \in Y = \Gamma \cup \Gamma_Q \\ U_e &\to S\#d \qquad \text{for } d \in \Delta_E \\ U_a &\to S\#d \qquad \text{for } d \in \Delta - \Delta_E \end{aligned}$$

Here, Δ_E contains all transitions that start from a state in Q_E.

Of course, the rules of the construction stage do not guarantee that v really encodes a consistent, accepting computation.

The target language contains all strings $d_i\$C_i \cdots \#d_0\$C_0\&w$, where d_i is chosen by ROMEO and cannot be applied to C_i. Hence, JULIET immediately wins if ROMEO chooses a wrong transition. Note that this part of the target language can be easily accepted by an NFA.

JULIET ends the construction stage whenever she wants by replacing S by U, resulting in a string v. Then the test stage starts and ROMEO has to check that v indeed encodes an accepting run of M. This is done as follows. First JULIET is required to select U. This is enforced by the fact that no strings in the target language start with the symbol U. On U, ROMEO has two possible rules, $U \to \bot$ or $U \to U'$.

ROMEO selects $U \to \bot$ if the string obtained so far is wrong in one of the following ways.

– Some transition chosen by JULIET cannot be applied.
– Some configuration is syntactically not of the right form, i.e., some C_i is either not of the form $(\Gamma(0+1)^N)^*\Gamma_Q(0+1)^N(\Gamma(0+1)^N)^*$ or not of the form $Y0^N(Y \cup \{0,1\})^*Y1^N$.
– C_t is not accepting.

It is easy to check with a finite automaton of size $O(N)$ whether the conditions above hold or not. The target language T contains all words starting with $\bot$ for which the string is in none of the above cases (ROMEO immediately looses if his objection if wrong). If the string is in one of the cases above, then JULIET looses in every possible continuation of the play, since all other strings included in T will start with U'.

In all other cases, ROMEO selects $U \to U'$. This rule, and all the following rules, has a navigation constraint $(\rightarrowtail, \epsilon, *)$, forcing JULIET to choose the symbols of the string consecutively.

It now remains for ROMEO to check that

(1) between consecutive #s (resp. between U and the first #, and between the last \$ and &), the binary counters are incremented successively from left to right, and
(2) for each i, configuration C_{i+1} follows from C_i via d_i or C_0 is not the initial configuration on input w.

During the rest of the game, ROMEO can choose for each symbol of the string, whether he wants to object or whether he leaves it as it is (rule $A \to A$). Whenever he objects, he replaces the symbols A with $\boxed{A,?}$ if (1) fails at the current position, and by $\boxed{A,!}$ if (2) fails at the current position.

We explain now how to deal with condition (1). Assume there is a substring $AuA'u'$ in v, where $A, A' \in Y$, $u, u' \in \{0,1\}^N$ but $\mathrm{bin}(u) + 1 \neq \mathrm{bin}(u')$, i.e., condition (1) is violated.

In this case, Romeo replaces A by an objection symbol indicating a counter mistake. In this situation, Juliet can only win, if she is able to show that the objection of Romeo is not valid. We describe next how this is done.

As we can assume that u and u' are of the same length N, $\mathrm{bin}(u) + 1 = \mathrm{bin}(u')$ holds, if and only if there is a position $0 \leq i < N$, such that

- $u[j] = u'[j]$ for all $j < i$,
- $u[i] = 0$ and $u'[i] = 1$,
- $u[j] = 1$, $u'[j] = 0$, for all $j > i$.

In order to win, Juliet must show that such a position i exists. Hence, after Romeo has rewritten A in $AuA'u'$ by the objection symbol $\boxed{A,?}$, Juliet jumps to some position i in u. We enforce this by associating the navigation constraints $(\rightarrowtail, (0+1)^*, 0)$ and $(\rightarrowtail, (0+1)^*, 1)$ with rules of the form $A \to \boxed{A,?}$. Romeo rewrites it via $0 \to \bar{0}$ and $1 \to \bar{1}$, respectively. Then Juliet successively has to select the remaining positions in u (which are all rewritten by $\bar{0}$ and $\bar{1}$, respectively with navigation constraint $(\rightarrowtail, \epsilon, *)$, i.e., go to the next position), and some initial positions of u' until Romeo selects another objection symbol via $0 \to \boxed{0,!}$ or $1 \to \boxed{1,!}$ at some position j of u'. Intuitively, Romeo claims that i and j witness that $\mathrm{bin}(u) + 1 \neq \mathrm{bin}(u')$ because one of the three conditions above fails.

Juliet wins, if i and j are no such witnesses, i.e., if the string between (and including) the two objection symbols is of one of the following forms, easily testable by the NFA T.

- $\boxed{A,?}(0+1)^{j-1}b(0+1)^{i-j-1}\bar{0}\bar{1}^*A'(\bar{0}+\bar{1})^{j-1}\boxed{b,!}$ for $A, A' \in Y$, $b \in \{0,1\}$ (case $j < i$),
- $\boxed{A,?}(0+1)^{i-1}\bar{0}\bar{1}^*A'(\bar{0}+\bar{1})^{i-1}\boxed{1,!}$ for $A, A' \in Y$ (case $j = i$),
- $\boxed{A,?}(0+1)^{i-1}\bar{0}\bar{1}^*A'(\bar{0}+\bar{1})^{i-1}\bar{1}\bar{0}^*\boxed{0,!}$ for $A, A' \in Y$ (case $j > i$).

Condition (2) can be handled in a simpler way. First, Romeo rewrites some $A \in Y$ by an objection symbol $\boxed{A,!}$. Then Juliet is forced to select successively all Y-positions in the next configuration (this can easily be enforced using navigation constraints) until Romeo rewrites some symbol B by $\boxed{B,!}$. Now, Juliet wins if either $\boxed{A,!}$ and $\boxed{B,!}$ are not followed by the same binary counter, or if A is consistent with B, the intermediate transition d, and the neighboring Y-symbols of B, or if the initial configuration is correct. To this end, the target automaton accepts a string if it contains a substring of one of the following kinds.

- $\boxed{A,!}(0+1)^i b_1(\Sigma \setminus \$)^*\$(\Sigma \setminus \$)^*\boxed{B,!}(0+1)^i b_2$ for $b_1, b_2 \in \{0,1\}$ with $b_1 \neq b_2$ (Romeo did not select the same position in the previous configuration),
- $\boxed{A,!}(\Sigma \setminus \#)^*\#d\$(\Sigma \setminus \$)^*C(0+1)^*\boxed{B,!}(0+1)^*D$ for $CBD \to_d A$ (if the position is the same, than the transition d was applied correctly),[5]
- $\boxed{A,!}\mathrm{bin}(i)(\Sigma \setminus (\# + \$))^*\&\Gamma_0^{i-1}\boxed{A,!}$ or $\boxed{(A, q_0),!}0^N(\Sigma \setminus (\# + \$))^*\&\boxed{A,!}$ for $i > 0$, $A \in \Gamma$ (the initial configuration is correct).

Finally, when the rewriting process reaches the symbol & this is rewritten by Romeo into \$. T accepts all strings in which the rightmost symbol which is not in the input alphabet of M is a \$. Hence, Juliet wins if Romeo never objects during the test stage. □

[5] Recall the definition of $CBD \to_d A$ from Subsection 2.2.

4.3 Deterministic games with unrestricted rules and left-to-right strategies

A *(reachability) pushdown game* is played by two players, ADAM and EVE, on a graph $G_{\mathcal{P}}$ induced by an alternating pushdown system $\mathcal{P} = \langle\, S = S_E \cup S_A, \Gamma, \delta, F \,\rangle$, i.e., a pushdown automaton without input, with existential and universal states. The nodes of $G_{\mathcal{P}}$ are the configurations of the automaton, i.e., elements from $S \times \Gamma^*$. To distinguish configurations of $\mathcal{P}$ from configurations of CF-games, we denote them in the form $[q, u]$ and refer to them as nodes of $G_{\mathcal{P}}$.

The set S of states is partitioned into existential (S_E, EVE's states) and universal (S_A, ADAM's states) states. Correspondingly, nodes $[q, u]$ are either existential or universal depending on whether $q \in S_E$ or $q \in S_A$. There is an edge from node $[q, u]$ to $[q', u']$ if $\mathcal{P}$ can go from $[q, u]$ to $[q', u']$ in one step, as described by the transition relation δ. A node $[q, u]$ is *final*, if $q \in F$.

EVE wins the reachability game from a given node $[q, u]$ if whatever ADAM's choices are, she can reach a final node. In this case, we call $[q, u]$ a *winning node*. It is known that deciding whether a node 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 relationship between pushdown games and CF-games with left-to-right strategies and DFA target language.

Proposition 4.4. *Given a CF-game $G = \langle\, \Sigma, R, T \,\rangle$, where T is a DFA with initial state q_0, one can construct in polynomial time a pushdown system $\mathcal{P}$ such that, for every string w,* JULIET *wins (G, w) with a left-to-right strategy if and only $[q_0, w\$]$ is a winning node of $\mathcal{P}$.*

Proof. Let $T = (Q, \Sigma, \delta_T, q_0, F_T)$. We define $\mathcal{P}$ as $(S = S_E \cup S_A, \Sigma \cup \{\$\}, \delta, \{f\})$, where

- $S_E = Q \cup \{f\}$, $S_A = \bar{Q}$ ($\bar{Q}$ is a disjoint copy of Q),
- for every pair $q \in Q$, $A \in \Sigma$ there are the transitions $\delta(q, A) = \{(\delta_T(q, A), \epsilon), (\bar{q}, A)\}$. A transition $(\delta_T(q, A), \epsilon) \in \delta(q, A)$ corresponds to the case where JULIET skips the current position, a transition $(\bar{q}, A) \in \delta(q, A)$ corresponds to the case where JULIET selects the current position,
- for every pair $(\bar{q}, A)$ and every rule $A \to u$ of G there is a transition $(q, u) \in \delta(\bar{q}, A)$. These transitions correspond to the selection of the corresponding rule in R by ROMEO, and
- finally, there is a transition $(f, \$) \in \delta(q, \$)$ for every accepting state q of T.

It can now be checked that JULIET wins (G, w) with a left-to-right strategy if and only if $[q_0, w\$]$ is a winning node in $\mathcal{P}$. For each node $[q, u\$]$, the state q is the state reached by T on the prefix processed so far in G, and u is the remaining suffix for G. □

Proposition 4.5. *Given a pushdown system $\mathcal{P}$ one can construct in polynomial time a game $G = \langle\, \Sigma, R, T \,\rangle$, where T is a DFA, such that any node $[q, A_1 \cdots A_n]$ of $\mathcal{P}$, c is winning if and only if* JULIET *wins the game G on $w = \boxed{q, A_1} A_2 \cdots A_n$ with a left-to-right strategy.*

Proof. Let $\mathcal{P} = \langle\, S = S_E \cup S_A, \Gamma, \delta, F \,\rangle$ and assume without loss of generality that the moves of $\mathcal{P}$ either pop a symbol or push one single symbol. We define a CF-game G with symmetric rule choice and navigation constraints capturing the behavior of the pushdown game $\mathcal{P}$.

In this game, nodes $[q, A_1 \cdots A_m]$ of $G_{\mathcal{P}}$ correspond to configurations $(u, \boxed{q, A_1} A_2 \cdots A_m)$ in the game G. Here u is a string of states of $\mathcal{P}$ and of special symbols $\sqcup$ and $\sqcap$ used as a kind of blank symbols. To simulate a step of $\mathcal{P}$ which pops the topmost symbol and enters

state r, the new configuration of G has to be of the form $(u', \boxed{r, A_2} A_3 \cdots A_m)$. Analogously, if $\mathcal{P}$ pushes a symbol B, the new configuration has to be of the form $(u', \boxed{r, B} A_1 \cdots A_m)$.

A pop move $(r, \epsilon) \in \delta(q, A)$ is simulated in two steps. First, a rule $\boxed{q, A_1} \to r$ with navigation constraint $(\rightarrowtail, \epsilon, *)$ (go to the right) is applied, then a rule $A_2 \to \boxed{r, A_2}$. We put $\boxed{q, A_1} \in \Sigma_E$ (JULIET selects the rule) if $q \in S_E$. If ROMEO selects a wrong rule, i.e., $A_2 \to \boxed{r', A_2}$ with $r' \neq r$, the target set lets JULIET win immediately, as it contains all strings of the form $ur\boxed{r', A}v$ with $r \neq r'$.

A push move $(r, B) \in \delta(q, A)$ is simulated by adding in G the rule $\boxed{q, A_1} \to \sqcup \boxed{r, B} A_1$. The $\sqcup$ symbol is here to prevent JULIET from immediately winning if the symbol to the left of $\boxed{q, A_1}$ was a state $q \neq r$ (recall from above that in the target language we have in the target language all strings of the form $ur\boxed{r', A}v$ with $r \neq r'$). We now have to make sure that JULIET selects next the symbol $\boxed{r, B}$ (recall that navigation constraints cannot enforce a move *inside* the right-hand side of the rule). This is done as follows. We add a rule $\sqcup \to \sqcap$ with a navigation constraint $(\rightarrowtail, \epsilon, *)$ (go to the right) and we force JULIET to choose immediately the symbol $\sqcup$ by making sure that no string in the target language contains the symbol $\sqcup$.

Furthermore, T accepts all strings of the form $u\boxed{f, A}v$ with $f \in F$, $A \in \Gamma$, $v \in \Gamma^*$ and $u \in (S \cup \{\sqcap\})^*$. Clearly, T can be accepted by a DFA of polynomial size in $\mathcal{P}$.

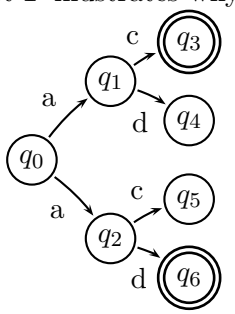
Now the statement of the proposition follows immediately. □

From Propositions 4.4 and 4.5 and [16, 5, 14] we obtain immediately the following result.

Theorem 4.6. *The combined decision problem for left-to-right strategies when the target language T is given by a DFA is* EXPTIME*-complete.*

Moreover, the set of input words for which JULIET *has a left-to-right winning strategy is regular and an alternating automaton recognizing it can be constructed in exponential time from a CF-game G.*

Remark 4.7. It follows from Theorem 4.3 that the upper bound in Theorem 4.6 does not apply in general to CF-games with a non-deterministic target automaton. The following non-deterministic target T illustrates why the proof of the upper bound in Proposition 4.4 fails.



In the game $G = \langle\, \Sigma, R, T \,\rangle$ where $R = \{b \to c \mid d\}$, JULIET has a winning strategy on ab: rewrite b, as both ac and ad are accepting for T. But in the pushdown system as constructed in the proof of Proposition 4.4, 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, ADAM will choose respectively d and c for replacing b and thus will end in a non-accepting configuration with state q_4 or q_5, respectively.

5 Non-recursive rules

In this section we focus on non-recursive games. The reader should recall the notion of depth for non-recursive games. For non-recursive games of depth d and width m the number of possible moves in a game is $O(m^{d+1}|w|)$. It is therefore straightforward that it can be checked in alternating exponential time, hence in ExpSpace, whether Juliet wins a game (G, w). If the depth bound d is considered fixed, $O(m^{d+1}|w|)$ becomes a polynomial bound resulting in a PSpaceupper bound. As each single non-recursive game has a fixed depth bound, the PSpaceupper bound holds also the data complexity of non-recursive games.

For unrestricted strategies, considered in Subsection 5.1, it turns out that all these upper bounds are tight.

In Subsection 5.2 left-to-right strategies are considered. Interestingly, in the bounded case, the restriction to left-to-right strategies does not reduce the complexity, it remains PSpace-complete. For non-recursive rules, the complexity is ExpTime.

Finally, we turn to deterministic games with non-recursive rules and left-to-right strategies in Subsection 5.3. The complexities are one level lower than for non-deterministic games, i.e., PSpace-complete for non-recursive games and PTime-complete for games with a fixed depth bound.

5.1 Non-recursive rules and unrestricted strategies

We first consider non-recursive sets of rules and unrestricted strategies. Again, the decision problems become decidable, although the complexity is high.

Theorem 5.1. *The combined decision problem for non-recursive CF-games is* ExpSpace-*complete.*

Proof. Both the lower and the upper bound use alternating exponential-time Turing machines.

We show the upper bound by constructing an ATM deciding whether Juliet has a winning strategy on w in exponential time. Let G be a game and w an input. The ATM maintains a string representing the current game configuration. At each step it checks whether this string is in the target language of the game. If yes, it stops and accepts. If not, it non-deterministically chooses a position to rewrite and universally branches over all possible rewritings. It is immediate to verify that this ATM is correct. As stated above, the maximal number of rule applications is $O(m^{d+1}|w|)$, if d is the depth and m the width of G. As each rule application replaces a single symbol by a string of length at most m the size of strings occurring in a play on w is $O(m^{d+2}|w|)$, which is also the time bound of the ATM.

We now prove the lower bound. Let M be an ATM working in time $2^{p(n)}$ on inputs of size n, for some polynomial p. We assume wlog that each state of M is either only entered from the left or only from the right. Let $w = a_1 \cdots a_n$ be an input of size n. We construct in polynomial time in w an extended CF-game and the string $w' = \$\boxed{q_0, a_1}S\$a_2S\$ \cdots S\$a_nS\$\bar{S}$ such that G is non-recursive and Juliet wins (G, w') if and only if M accepts w. The construction is similar as in the proof of Lemma 4.1. However, in that proof a symbol representing a tape cell could be rewritten arbitrarily often, which is not allowed for a non-recursive game. The solution here is to produce for each tape cell a substring of the form $\$\sqcup^{2^{p(n)}}$, that will allow to store the consecutive symbols in that cell. During the simulation of the ATM by the game,

the symbols $\sqcup$ might be replaced by symbols from Γ, i.e., the tape alphabet of M, in a left-to-right manner. A string $uA\sqcup^i$, $u \in \Gamma^*, A \in \Gamma$, represents a tape cell containing A. Each $\sqcup$ can be rewritten only once.

In the first phase, G has the rules $S \to S_1S_1$, $S_i \to S_{i+1}S_{i+1}$, for each $i < p(n)$, and $S_{p(n)} \to \sqcup$, that allow to derive a string $\sqcup^{2^{p(n)}}$ from each S in w'. In a similar fashion, from $\bar{S}$ a string $(\sqcup^{2^{p(n)}}\$)^{2^{p(n)}-n}$ can be derived. JULIET is responsible that the first phase is performed as intended, otherwise she might loose in the second phase. The target set T_1 is empty, and the set W_1 allowing to continue with the second phase ensures that all occurrences of $S, \bar{S}$ have been replaced by the $\sqcup$'s. In phase 2, JULIET first has to select position 2.

The sequence of game configurations corresponding to a transition $(q, A, r, B, \text{Right})$ of M is the following (v and w are strings that do not contain \$):

$$\begin{aligned}(u\$v, \boxed{q, A}\sqcup^i \$wC\sqcup^j \$x) &\to (u\$v\boxed{r, B, \to}\sqcup^i \$w, C\sqcup^j \$x)\\ &\to (u\$v\boxed{r, B, \to}, \sqcup^i\$w\boxed{r, C}\sqcup^j \$x)\\ &\to (u\$v\boxed{r, B, \to}B\sqcup^{i-1} \$w, \boxed{r, C}\sqcup^j \$x)\end{aligned}$$

This can be enforced using the following rules with constraints:

- $\boxed{q, A} \to \boxed{r, B, \to}$ with constraints "go right until you pass a \$ and then go to the last symbol different from $\sqcup$";
- $C \to \boxed{r, C}$ with constraints "go left until you pass a \$ and then go to the leftmost $\sqcup$ symbol"[6];
- $\sqcup \to B$ with constraints "go to the right to the next symbol of the form $\boxed{\sigma}$", with $\sigma \in Q \times \Gamma$;

The case of left moves is similar. The symbols for which JULIET chooses the rewriting rule are symbols $\boxed{q, A}$, where q is an existential state.

It is easy to adapt the target language of Lemma 4.1 in order to enforce that ROMEO always picks the correct rewriting rule as in the sequence described above.

Notice that we do not literally use the navigation constraints of Section 3 but slightly more general ones. They allow to specify also the neighbor of the new position in the opposite direction from the old position (e.g., if the navigation constraint forces to go right, then we have a constraint on the right neighbor of the target position). It is immediate to extend the techniques of Section 3 in order to allow such constraints. □

We now consider games for which the depth is bounded by some fixed integer d.

Lemma 5.2. *There is an extended CF-game $G = \langle \Sigma, R, T\rangle$ of depth 1 such that Data(G) is* PSPACE*-hard.*

Proof. We use a reduction from the quantified Boolean satisfaction problem QBS. The input of QBS is a formula $\Phi = \exists x_1 \forall x_2 \cdots \forall x_n \varphi$ where φ is a CNF formula with m clauses and 3 literals per clause. We assume here, wlog, that the quantification starts with an existentially quantified variable, alternates for each quantifier and ends with a universally quantified variable.

The extended game we construct has two phases and is played on a straightforward string encoding of Φ. Note that it is necessary to encode the variables of Φ, as the game has to

[6] Because of the transition $(q, A, r, B, \text{Right})$, state r can be entered only from the left.

be the same for all possible formulas. Basically, in the first phase, all variables get a truth value, alternatively chosen by JULIET and ROMEO, then ROMEO selects a clause and JULIET a literal in the clause. In phase 2 it is checked that this literal evaluates to true. We encode every variable x_i by the string a^i, where a is a symbol. For each clause C_i let w_i be the string $\#\#_{\delta_1} b^{j_1} \#_{\delta_2} b^{j_2} \#_{\delta_3} b^{j_3}$, where $j_1, j_2, j_3, \delta_1, \delta_2, \delta_3$ are such that $x_{j_1}, x_{j_2}, x_{j_3}$ are the variables occurring in C_i and δ_k is 0 if x_{j_k} occurs negatively, 1 otherwise.

Consequently, we set $w = \# \exists a \# \forall a^2 \# \cdots \# \forall a^n \# \$w_1 \cdots w_m\$$.

We now describe the first phase of the game more formally.

JULIET has to choose position 2 first. The rules for the first part of the first phase are $\forall \to 0$, $\forall \to 1$, $\exists \to 0$ and $\exists \to 1$. The symbol $\exists$ is in $\Sigma_{E,1}$ hence JULIET chooses the right-hand side. Each rule has navigation restrictions $(\rightarrowtail, a^*\#, \exists)$, $(\rightarrowtail, a^*\#, \forall)$ and $(\rightarrowtail, a^*\#\$, \#)$, hence JULIET can select the next variable to the right or, if all variables have been replaced, the first symbol # of w_1.

The rules for the second part of the first phase are

- $\# \to s$ and $\# \to r$: Hence, ROMEO can select or reject a clause. The rule $\# \to r$ has a navigation constraint $(\rightarrowtail, (\Sigma - \#)^*, \#)$ which forces JULIET to go to the next symbol # to the right. The rule $\# \to s$ has navigation constraints $(\rightarrowtail, (\Sigma - \#)^*, \#_0)$ and $(\rightarrowtail, (\Sigma - \#)^*, \#_1)$. This forces JULIET to select a variable in the current clause;
- $\#_0 \to 0$ and $\#_1 \to 1$: The selected variable is marked as selected. These rules have navigation constraints which force JULIET to end phase 1 and to proceed by phase 2.

In order to prevent ROMEO from selecting two or zero clauses, we take $T_1 = \Sigma^* s \Sigma^* s \Sigma^* + (\Sigma - s)^* r (\Sigma - \#_2)^*$.

We now describe the second phase. During this phase, it is verified that the literal which JULIET selected in the clause chosen by ROMEO becomes true by the value assigned to its variable. To this end, JULIET selects a string of a's in the first part of w. Then it is checked that the number of these as is the same as the number of bs at the selected literal. This check is performed by replacing alternatively each a by c and each b by c. Finally, it is checked that at the left of the as there is a 1 if and only if there is a 1 to the left of the bs. The rules are as follows.

- $a \to c$: The as are replaced from right to left, whereas the bs are replaced from left to right. Therefore, the associated navigation constraint is $(\rightarrowtail, c^*\#\Sigma^* s \Sigma^* (0+1) c^*, b)$.
- $b \to c$: The navigation constraint to get back is $(\leftarrowtail, c^* c \# \Sigma^* s \Sigma^* (0+1) c^*, a)$. In order, to be able to get to the first \$ after the comparison is finished, there is another navigation constraint $(\leftarrowtail, \Sigma^*, \$)$.
- Finally, there is a rule $\$ \to \$'$ which ends the game.

The target language is reached for JULIET if the final string is in

$$(\Sigma^* 0 c c^* \# \Sigma^* \$' \Sigma^* 0 c c^* (\#_0 + \#_1 + \# + \$) \Sigma^*) + (\Sigma^* 1 c c^* \# \Sigma^* \$' \Sigma^* 1 c c^* (\#_0 + \#_1 + \# + \$) \Sigma^*).$$

□

When d is fixed, $O(m^{d+2}|w|)$ becomes a polynomial bound. In that case, the upper bound proof of Theorem 5.1 gives an PSPACE upper bound. Together with Lemmas 5.2, 3.1 and 3.4 for the lower bound, this yields the following statement:

Theorem 5.3. *For each $d \geq 1$, the combined decision problem for d-bounded CF-games is* PSPACE*-complete. There exists a fixed game G of depth 1 for which Data(G) is* PSPACE*-complete.*

5.2 Non-recursive rules and left-to-right strategies

We continue with non-recursive rules, but we now concentrate on left-to-right strategies. It turns out that the complexity of the combined decision problem for bounded games is the same as in the case of unrestricted strategies. Recall that the data decision problems are all regular for left-to-right strategies.

The following lemma considers games that are bounded *and* unary. It will be used again in Section 6 to prove Theorem 6.4.

Lemma 5.4. *For each $d \geq 1$, the combined decision problem for extended CF-games that are d-bounded and unary with left-to-right strategies is* PSPACE*-hard.*

Proof. The proof is again a reduction from the quantified Boolean satisfaction problem QBS. As in Lemma 5.2, we first construct a game with symmetric rule choice and navigation constraints. As we are dealing with combined complexity here, the construction is even simpler than the one of Lemma 5.2, because we are able to use one symbol per variable as opposed to encoding them.

For a given QBS formula $\exists x_1 \forall x_2 \cdots \forall x_n \varphi$ where φ has m clauses of 3 literals each, let w be the string $\exists x_1 \forall x_2 \cdots \forall x_n \# u_1 \cdots u_m$. Here, u_i is of the form $(S v_1 v_2 v_3)$ with $v_j = 1x_k$ if the j-th literal of C_i is x_k and $v_j = 0x_k$ if it is $\neg x_k$.

In the game, first a value is associated to each variable, by JULIET for the existentially quantified variables and by ROMEO for the others. For this we have rules $\exists \to 0$, $\exists \to 1$, $\forall \to 0$ and $\forall \to 1$. Next, ROMEO decides for each clause whether to accept or reject it ($S \to a$, $S \to r$). The regular expression T checks in a straightforward manner whether at least one of the literals of the selected clause becomes true by the variable assignment. □

From Lemmas 5.4, 3.2 (for the lower bound) and an immediate adaptation of the proof of Theorem 5.1 for left-to-right strategies (for the upper bound) we obtain:

Theorem 5.5. *For each $d \geq 1$, the combined decision problem for d-bounded CF-games and left-to-right strategies is* PSPACE*-complete.*

For non-recursive rules the complexity is one level higher.

Theorem 5.6. *The combined decision problem for non-recursive games and left-to-right strategies is* EXPTIME*-complete.*

Proof. Both the upper and lower bound proof use ATM of polynomial space.

The upper bound is shown similarly as in Theorem 5.8 below. The only differences are that the number of pointers that we store is polynomial (bounded by the size of the rule set) and that the target is an NFA (hence, we need to store a set of states instead of the state pointer).

We now prove the lower bound. Let M be an ATM with polynomial space bound $p(n)$ and time bound $2^{p(n)}$ on inputs of size n. We assume wlog that M strictly alternates between existential and universal states. Let Q be the set of states of M. Let w be an input of size n. We construct in polynomial time an extended CF-game $G = \langle \Sigma, R, T \rangle$ such that G is non-recursive and JULIET has a winning left-to-right strategy in G on $w\$S_1$ iff M accepts w. The extended game uses symmetric rule choice, only. During the game JULIET and ROMEO essentially derive the sequence of configurations of M: JULIET produces the existential configurations while ROMEO produces the universal ones.

This is done as follows. Let Γ be the tape alphabet of M. Let $\hat{\Gamma}_e = \{\boxed{A,q} \mid A \in \Gamma, q \text{ an existential state of } Q\}$, $\hat{\Gamma}_a = \{\boxed{A,q} \mid A \in \Gamma, q \text{ an universal state of } Q\}$ and $\hat{\Gamma} = \hat{\Gamma}_e \cup \hat{\Gamma}_a$. The configuration slots are produced by the rules $S_{i-1} \to S_i S_i$ for each $1 < i < p(n)$ and $S_{p(n)-1} \to \#C_1^e\$C_1^a$, eventually resulting in a string of $2^{p(n)-1}$ copies of $\#C_1^e\$C_1^a$. The configurations are generated using rules $C_i^a \to D_i^a$, $D_i^a \to AC_{i+1}^a$, $C_i^e \to D_i^e$ and $D_i^e \to AC_{i+1}^e$ for all $i < p(n)$ and all $A \in \Gamma \cup \hat{\Gamma}$. To allow a player to object immediately, if the other produces a symbol which leads to a configuration that is not consistent with the previous one, we add the rules $C_i^a \to \boxed{!}$ and $C_i^e \to \boxed{!}$. The symbols C_i^a and D_i^e belong to Juliet, hence, she chooses configurations from C_1^e (via D_i^e) and can object choices that Romeo makes on D_1^a (via C_i^a).

Each configuration should contain exactly one symbol from $\hat{\Gamma}$ which represents the position of the head and the current state of M. Let $Y = \Gamma \cup \hat{\Gamma}$.

For each $t = (A, B, C, D) \in Y^4$ let R_t be the expression $\cup_{j \le p(n)} \Sigma^* \# Y^{j-2} ABCY^* \$ Y^{j-1} D\boxed{!}$. This expression checks that the symbol after which a player objected is D, the same position carried a B in the configuration before with left and right neighbors A and C, respectively. A 4-tuple t is said to be *existential* (resp. *universal*) if it is actually in $(\Gamma \cup \hat{\Gamma}_E)^4$ (resp. in $(\Gamma \cup \hat{\Gamma}_V)^4$).

The target language contains all strings with a substring in which the last symbol is $\boxed{!}$ reflecting one of the following situations.

- Romeo is currently producing a configuration, and the last symbol before $\boxed{!}$ shows that this configuration is inconsistent with the previous configuration. Such inconsistencies are accepted if the string conforms to some R_t where $t = (A, B, C, D)$ is universal but $ABC \not\to_M D$.
- Juliet is currently producing a configuration, and the last symbol before $\boxed{!}$ shows that this configuration is consistent with the previous configuration. This is captured by the expressions R_t where $t = (A, B, C, D)$ is existential and $ABC \to_M D$.
- Juliet is currently producing the initial configuration, Romeo objected, but the position before $\boxed{!}$ is consistent with the input string. This can be checked in a similar as consistency between the configurations.

The proof is finished by applying Lemma 3.2. □

5.3 Deterministic games with non-recursive rules and left-to-right strategies

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

We will need the following lemma which will be also used later in the proof of Theorem 6.6.

Lemma 5.7. *The combined decision problem for d-bounded and unary extended CF-games with DFA target language and left-to-right strategies is hard for* PTime.

Proof. The proof is by a reduction from the monotone Boolean circuit value problem [13]. Let C be a Boolean circuit with 0s and 1s at the input gates. We can assume wlog that all paths in C are alternating between **or** and **and** gates, have fan-in two and, start and end

with **and** gates [13]. With C we associate a DFA T as follows. The states of T are the gates of C. The input alphabet for T is $\{l, r\}$. Strings over this alphabet describe paths in C starting from the output gate. The letter l corresponds to the left input of a gate, the letter r to the right one. We define $\delta(g, l) = g'$, if g' is the left input gate for g and, $\delta(g, r) = g'$ if it is the right input. For input gates g, we set $\delta(g, l) = \delta(g, r) = g$. The final states of T are the input gates with value 1. Hence, T accepts exactly those strings that have a prefix corresponding to a path from the output gate to an input gate with value 1.

In the game JULIET tries to prove that g evaluates to 1 and ROMEO tries to show it evaluates to 0. They define a path from the output gate to an input gate and JULIET wins if this input gate has value 1. The game simply alternates between ROMEO-moves (at and-gates) and JULIET-moves (at or-gates). To this end, the initial string w is $(S_A S_E)^n$, where $2n$ is the depth of C and $S_E \in \Sigma_E$, $S_A \in \Sigma_A$. The rules are $S_E \to l \mid r$ and $S_A \to l \mid r$ with navigation constraint $(\rightarrowtail, \epsilon, \star)$ (go to the right).

It is straightforward to verify that JULIET has a winning strategy iff the circuit evaluates to 1 and that the reduction is logarithmic space (actually first-order).

□

Theorem 5.8. *For each d, the combined decision problem for d-bounded CF-games with left-to-right strategy and DFA target language is* PTIME*-complete.*

Proof. The lower bound follows from Lemmas 5.7, 3.2 and 3.4.

We prove the upper bound by constructing an ATM M which decides in logarithmic space whether JULIET has a winning left-to-right strategy for the game $G = \langle \Sigma, R, T \rangle$ on a string w. The tape of M consists of $d + 2$ pointers to its input. The first one (the *string pointer*) points to the letter it is currently processing in w. The second one (the *state pointer*) points to a state of the DFA for T. Each of the others (the *stack pointers*) point to a position in the right-hand-side of a rule of R currently being investigated. Initially the string pointer is set to the first letter of w, the state pointer to q_0 and all other pointers are set to null.

At each step the machine checks whether the current situation of M represents a configuration of G in which JULIET has won. This can be done by starting T from the state currently pointed at, and reading the string $a_k v_k v_{k-1} \cdots v_1 w'$, where

- w' is the remainder of w starting behind the string pointer,
- $k \leq d$ is the number of non-null stack pointers and also the current depth, and
- $u_i a_i v_i$, for each $i \leq k$, is the right-hand side of the rule of the i-th stack pointer, which currently points at the position of a_i.

If yes it immediately stops and accepts. Otherwise, it reads the letter a_k and non-deterministically decides whether it rewrites this letter or skips it. To rewrite it, it universally branches over all possible rules and sets the lowest null stack pointer to the first position of the right-hand side of the chosen rule. To skip it, it moves the stack pointer k to the next symbol and updates the state pointer according to a_k. Whenever a stack pointer k aims to move to the right from the last symbol of its rule, it is reset to null and the algorithm continues with stack pointer $k - 1$. Since d is a constant, the tape size remains logarithmic. □

If the depth of the rule set is not fixed, the complexity is one level higher:

Theorem 5.9. *The combined decision problem for non-recursive games with DFA target and left-to-right strategies is* PSPACE*-complete.*

Proof. The lower bound is obtained by a reduction from QBS. Let φ be a quantified Boolean formula in prenex form, with quantifier-free part in CNF, clauses $C_1, \ldots, C_m$ and variables $x_1, \ldots, x_n$. We encode φ by the string $v = \exists x_1 \forall x_2 \cdots \forall x_n \# u_1 \cdots u_m$, where u_i is of the form $(v_1 v_2 v_3)$ with $v_j = 1x_k$ if the j-th literal of C_i is x_k and $v_j = 0x_k$ if it is $\neg x_k$. Hence, each variable x_i of φ is represented by its own symbol. The game is played on the string $w = S_1 v$. From S_1 a variable assignment is constructed in a right-to-left manner. For this purpose we have rules $S_i \to S_{i+1} x_i 0$ and $S_i \to S_{i+1} x_i 1$, for each $i < n$, as well as $S_n \to \$x_n 0$ and $S_n \to \$x_n 1$. A symbol S_i is in Σ_E if and only if x_i is existentially quantified. From \$ we let Romeo choose a clause index j and Juliet choose a variable index i. I.e., there are rules $\$ \to \& j$, for each $j \leq m$, and $\& \to x_i$, for each $i \leq n$. A DFA of polynomial size can check on input $x_i \, j \, x_n a_n \cdots x_1 a_1$ that the variable assignment $x_i \mapsto a_i$ makes clause C_j true. That is, either $a_i = 1$ and $1x_i$ occurs between the jth pair of brackets, or $a_i = 0$ and $0x_i$ occurs.

For the upper bound we describe an alternating polynomial time algorithm. Let $G = \langle \Sigma, R, T \rangle$ be a game with target DFA $T = (\Sigma, Q, \delta, q_0, F)$.

For a string w, a state $q \in Q$ and a set $X \subseteq Q$ of states, we write $q \Rightarrow_J^w X$ if Juliet has a strategy that guarantees that the string w' that is obtained at the end of the game, satisfies $\delta^*(q, w') \in X$. Clearly, Juliet has a winning strategy on w in the CF-game G if and only if $q_0 \Rightarrow_J^w F$.

It turns out that whether $q \Rightarrow_J^w X$ holds can be computed in a recursive fashion in alternating polynomial time, using the following two properties of $\Rightarrow_J^w$.

(a) $q \Rightarrow_J^{uv} X$ if and only if there is a set $Z \subseteq Q$ such that $q \Rightarrow_J^u Z$ and, for all $p \in Z$, $p \Rightarrow_J^v X$.
(b) If $B \to u_1 \mid \cdots \mid u_m$ are all rules with left-hand side B, then $q \Rightarrow_J^B X$ if and only if, $\delta(q, B) \in X$ or, for all $i \leq m$, $q \Rightarrow_J^{u_i} X$.

To see that (a) holds, assume that $q \Rightarrow_J^{uv} X$ and fix a winning strategy for Juliet on uv. Let U be the set of strings that Romeo can reach by playing the game on u given the strategy of Juliet. Let $Z = \{\delta^*(q, u') \mid u' \in U\}$. Hence, $q \Rightarrow_J^u Z$ holds. On the other hand, as Juliet's strategy on uv guarantees to end up in a state of X, for every state $p \in Z$, Juliet can play such that for the string v' evolving from v it holds $\delta^*(p, v') \in X$. The reverse direction is straightforward. Property (b) follows directly from the definition, as Juliet has no other choice than to select B or to stop immediately.

The alternating polynomial time algorithm works as follows. On input q, w, X, with $|w| > 1$ it guesses a set Z and non-empty strings u, v with $w = uv$, and it checks recursively that $q \Rightarrow_J^u Z$ holds and, for all $p \in Z$, $p \Rightarrow_J^v X$. On input q, B, X it checks whether $\delta(q, B) \in X$ or, for all right-hand sides u of rules for B, that $q \Rightarrow_J^u X$. Each computation path has length at most $|w| + m|\Sigma|$, where m is the width of R, since G is non-recursive. □

6 Linear rules

In this section we focus on linear and unary games. Recall from our example in the introduction that in practice, service calls often generate a single subsequent call, a setting which is modeled by linear CFGs in our framework.

A crucial property of plays on linear games, which is particularly important for some of the upper bounds, is that the number of non-terminal symbols never increases. Hence, starting from an initial string w of length n, all strings occurring during a play are of the form $u_0 A_1 u_1 \cdots u_{k-1} A_k u_k$, for some $k \leq n$. The behavior of the target automaton on the terminal strings u_i can be subsumed in *behavior relations* $f_i \subseteq Q \times Q$. Therefore, a configuration can

be represented always by a string of length $O(n|Q|^2)$. If T is deterministic, the f_i are actually functions, which is important for some of the proofs.

In Subsection 6.1 we consider unrestricted strategies. In all cases, combined or data complexity, linear or unary rules, we obtain the same complexity, ExpTime-complete.

Subsection 6.2 shows that also for left-to-right strategies there is no complexity difference between unary and linear rules. It is PSpace-complete for both, therefore one complexity level below the case of unrestricted strategies.

Only for deterministic games unary and linear rules are different as shown in Subsection 6.3. Whereas the complexity remains PSpace-complete in the linear case, unary rules yield another tractable case, PTime-complete.

6.1 Linear rules and unrestricted strategies

We start with the following lower bound which implies the same lower bound for combined decision problems of unary and linear games.

Lemma 6.1. *There is a unary CF-game G, for which Data(G) is* ExpTime*-hard.*

Proof. Let L be an ExpTime-complete language. With a classical padding argument, we can find such a language that is accepted by an ATM M using space n on inputs of length n.

From Lemma 4.1 we know that there is a unary CF-game G such that for every input $w = a_1 \cdots a_n$ to M, Juliet wins (G, w'), where

$$w' = \$\boxed{q_0, a_1}\, a_2 \cdots a_n \#,$$

if and only if M accepts w. The CF-game G has the desired property. □

The following lemma shows that the combined decision problem for *linear* games is in ExpTime.

Lemma 6.2. *The combined decision problem for linear CF-games is in* ExpTime*.*

Proof. Let $G = \langle \Sigma, R, T \rangle$ and let w be the initial string. Let n be the number of non-terminal symbols occurring in w. By linearity of R this is an upper bound on the number of non-terminal symbols during the whole game. Let Q be the set of states of T. The string w can be written as $u_0 A_1 u_1 A_2 \cdots A_n u_k$ where the A_i are non-terminal letters and the u_i are words over the terminal alphabet of R. Each u_i induces a transition relation $f_i \subset Q \times Q$ which can be coded by $|Q|^2$ symbols. We now define an alternating polynomial space Turing machine M that decides whether Juliet has a winning strategy. M maintains on its tape a string of the form $f_0 B_1 f_1 \cdots B_k f_k$ with $k \leq n$. This requires space of $O(k|Q|^2)$. Whenever the composition of the $k+1$ transition relations with the behavior of T on the intermediate non-terminals gives an accepting state of T, M stops and accepts. Otherwise it decides non-deterministically which non-terminal it is going to rewrite, say B_i. It then starts universally a new computation for each rule in R whose head is B_i. Each subcomputation, corresponding to a rule $B_i \to \alpha B \beta$, where B is the only non-terminal, starts by computing the transition relation for α and β, and composes them with f_{i-1} and f_i, respectively. Altogether this is an alternating polynomial space computation and thus in ExpTime. □

By combining Lemma 6.2 with Lemma 6.1, we get the following theorem.

Theorem 6.3. *For unary and linear games with unrestricted strategies the combined complexity is* ExpTime*-complete. There are games, for which the data complexity is* ExpTime*-complete.*

6.2 Linear rules and left-to-right strategies

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

Theorem 6.4. *The combined decision problem for linear and for unary CF-games with left-to-right strategies is* PSPACE*-complete.*

Proof. The lower bound follows from Lemmas 5.4 and 3.2. For the upper bound it is enough to consider the linear case. Let $G = \langle \Sigma, R, T \rangle$ and $w \in \Sigma^*$. We construct a non-deterministic Turing machine M that checks in polynomial space whether ROMEO has a winning strategy in G on w. The result then follows because NPSPACE=PSPACE. M guesses the strategy of ROMEO and does backtracking for the possible moves of JULIET. For linear rules these possible moves are either to stay at the current non-terminal or to jump to the next one. Note that we consider here the pure game, where only ROMEO selects rules.

As in the proof of Lemma 6.2, w is transformed into $f_0 A_1 \cdots f_{k-1} A_k f_k$ where the A_i are the non-terminals of w and the f_i are the corresponding intermediate transition relations of T. Together with the current configuration of the game, M will maintain a pointer to the non-terminal it is currently investigating and a counter m_i, for each $i \leq k$, in order to check that all choices of JULIET have been considered.

M works as follows. It cycles through all possible strategies of JULIET. It has to verify that during all plays *never* a string in the target language is produced. For each non-terminal it first considers jumping to the next non-terminal or staying at the same one, in a backtracking fashion. In either case M guesses the rule choice of ROMEO, updates the current configuration of the game (transition relations and non-terminal symbol). If M stays with the same non-terminal it increments the counter m_i of the changed position by one. If the counter goes beyond $|\Sigma| \times 2^{2(|T|)^2}$, M can safely backtrack because some possible outcome at this position (wrt the target T) has already occurred twice. Note that $2^{2(|T|)^2}$ is an upper bound for the number of different possible behavior relations for T. □

6.3 Deterministic games with linear rules and left-to-right strategies

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

Theorem 6.5. *The combined decision problem for linear CF-games with target DFA and left-to-right strategies is* PSPACE*-complete.*

Proof. The upper bound follows from Theorem 6.4.

For the lower bound, let M be a deterministic Turing machine that works in space bounded by a polynomial p. Let Γ_0 be the input alphabet, Γ the tape alphabet, Q its set of states and $w \in \Gamma_0^*$ be an input of size n. We can assume wlog that M halts on every input. We construct in polynomial time a game $G = \langle \Sigma, R, T \rangle$ where R is linear and T is a DFA such that JULIET has a winning strategy in G on $S\#q_0 w \sqcup^{p(n)-n} \#$ iff M accepts w. The idea of the construction is as follows. Using S, JULIET lets ROMEO produce a string which is intended to represent the computation of M on w. JULIET wins iff either ROMEO does not produce a computation or all, or if it is accepting. The string will be a sequence $C_l \# C_{l-1} \cdots \# C_0$ of configurations of M. If ROMEO does a mistake JULIET wins because the DFA T accepts the current string. To this end, it checks the mistake by counting and comparing the corresponding positions.

More precisely, $\Sigma = \Gamma \cup Q \cup \{\#, S\}$, the rules of G are $S \to Sa$ for every $a \in \Sigma \setminus \{S\}$. The target T accepts all words such that the distance between two successive #s is not $p(n)$ (that can be checked deterministically with $O(p(n))$ states). It also accepts all words in which C_l is an accepting configuration of M, and those containing a substring between successive occurrences of # which does not encode a configuration of M.

To capture inconsistent configurations produced by ROMEO immediately, it finally contains all words with a mismatch between the first letter and the previous configuration of M, whose description starts after the first # symbol. The latter can be done by a deterministic automaton of size polynomial in $|M| + |w|$ as follows: first it computes, using less than $p(n)$ states, the distance d between the first letter, say D, and the first # symbol, then it reads, and remembers in its state the state q of the previous configuration of M. Then, it skips the next $p(n) - d - 1$ characters and accepts if for the following three symbols A, B, C it does not hold $ABC \to_M D$. All this requires only a polynomial number of states. □

Theorem 6.6. *The combined decision problem for unary CF-games with target DFA and left-to-right strategies is* PTIME*-complete.*

Proof. The lower bound follows from Lemma 5.7 and Lemma 3.2.

We now prove the upper bound. We construct an alternating Turing machine deciding in logarithmic space whether JULIET has a winning left-to-right strategy. Let G be a unary game with T as a target DFA. The Turing machine maintains on its tape a pointer to the current state in the target automaton and a pointer to the current letter in the input configuration. It also has a counter of logarithmic size. At each step it first checks whether the current configuration is in the target language. If yes it immediately stops and accepts. Otherwise, it non-deterministically decides whether it rewrites the current letter or not. If not, it updates its state pointer by simulating T with the current letter and the current value of the state pointer. Then it resets the counter value to zero and moves to the next letter. Whenever it reaches the end of the string it rejects. To rewrite it universally branches over all possible rewritings. If the newly inserted symbol is a non-terminal, it increments the counter by one. If the counter goes beyond $|\Sigma|$ then all possible cases for the current position have been investigated and the Turing machine stops and rejects. If the newly inserted symbols is a string of terminal symbols then it updates the state pointer by simulating T on that string and proceeds with the next symbol. □

7 Rules with regular expressions

As mentioned in the introduction the initial motivation of this work was to consider *extended* context-free rules (ECFG), i.e., rules of the form $A \to R_A$ where R_A is a regular expression. We first consider unrestricted strategies and show that even the data complexity of the *non-recursive* case is undecidable. However for left-to-right strategies, we show that the general decidability result of the previous section extends to these games.

Extended rules, unrestricted strategies

Theorem 7.1. *The data decision problem for 2-bounded CF-games with extended context-free rules is undecidable.*

Proof. We give a reduction from the problem of whether a (deterministic) Turing machine M accepts the empty word or not. Then ROMEO wins the game iff M accepts ϵ. The game starts with a unique letter S. The idea is to simulate M as in the proof of Theorem 5.1 but reversing the roles of JULIET and ROMEO. This can be done because M is deterministic and therefore JULIET has no decision to take in the simulation of M. The first rule of the game is $S \to (\$\sqcup^*)^*$ allowing ROMEO to introduce at once enough space for M to accept ϵ. If M requires m steps to accept ϵ then ROMEO rewrites S with $(\$\sqcup^m)^m$ in one step. Then, the game continues by a simulation of M on ϵ, as in the second part of the proof of Theorem 5.1. Note that the simulation is achieved with rules of depth one, and that we do not need symmetric rules for simulating M because it is deterministic. The target language contains all strings corresponding to the case where the space provided by ROMEO does not suffice for completing the run of M (i.e., the current configuration allows a further step, but there is no extra $\sqcup$ symbol available) and to the case where the run is blocked in a non-final configuration.

We now apply Lemmas 3.1, 3.4 to the second step (simulation of M), obtaining a game of depth one. Together with the first step this yields the overall depth 2.

□

Extended rules, left-to-right strategies

The case of CF-games was handled by a reduction to *reachability* pushdown games. In order to extend this idea to ECFGs we need to consider more powerful pushdown games: *parity* pushdown games.

A *parity pushdown game* is played on a graph $G_{\mathcal{P}}$ associated with an alternating pushdown system $\mathcal{P} = \langle Q = Q_E \cup Q_A, \Gamma, \delta, W \rangle$, where Q, Γ, δ are defined as in the reachability pushdown game (see Section 4). In addition, W is a function that associates an integer (priority) with each state of Q.

A play π is a (possibly infinite) sequence of states that corresponds to the choices made by EVE and ADAM starting from the initial node. If the play is finite, one of the player cannot play anymore, then the other player wins immediately. If the play is infinite, consider the sequence $W(\pi)$ and consider the lowest number n which is repeated infinitely often in $W(\pi)$. EVE wins the play if this number is even, otherwise ADAM wins.

We say EVE *wins the parity game* if she has a winning strategy. It is known that deciding whether an initial 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].

Proposition 7.2. *Given a game $G = \langle \Sigma, R, T \rangle$, where T is a DFA with initial state q_0 and a set R of extended context-free rules, one can construct in polynomial time a parity pushdown system $\mathcal{P}$ such that* JULIET *wins the game G on w if and only if the node $[q_0, w\$]$ is winning for* EVE *in the parity pushdown game.*

Proof. Starting from G, we first construct a (pure) CF-game $G' = \langle \Sigma', R', T' \rangle$ that in a sense approximates G. Then we apply the reduction of Proposition 4.4 to obtain a pushdown system for G'. We then show how to define the accepting configuration of G' is order to simulate G appropriately.

We construct $G' = \langle \Sigma', R', T' \rangle$ as follows. For each extended rule $A \to R_A$ of G, let $\Gamma(R_A)$ be a right-linear context-free grammar generating R_A where each non-terminal letter is a new letter (not from Σ). Let Δ be the set of new symbols introduced in $\Gamma(R_A)$. Let R' be the set

of all rules that are either in $\Gamma(R_A)$ or of the form $A \to \bot$ (ROMEO also has the possibility to object using $\bot$). The target language is T. Note that T does not contain any symbol from Δ, thus JULIET has to rewrite them if she wants to win. Note also that because of the left-to-right strategy JULIET is forced to select the leftmost symbol of Δ if there is one.

So G' simulates G by simulating a rule $A \to R_A$ of G by a sequence of rewritings in G' using the rules of $\Gamma(R_A)$. The problem is that ROMEO can now win in G' by refusing to complete a simulation of $A \to R_A$ and rewriting forever using the rules of $\Gamma(R_A)$ forcing an infinite play. We prevent him from doing that by adding an appropriate parity condition in the pushdown system constructed from G' in the proof of Proposition 4.4.

Let $\mathcal{P}$ be the pushdown system constructed from G' as in Proposition 4.4. We modify $\mathcal{P}$ and define the parity winning condition as follows. All nodes with an accepting state (corresponding to a word in T') have priority 0, are self-looping and belong to EVE (EVE safely wins if she gets there). All nodes corresponding to a symbol of Δ have priority 2: EVE wins if ADAM keeps rewriting forever. The remaining nodes have priority 1: ADAM wins if he can force an infinite play which is "fair" (no infinite rewriting when simulating a rule $A \to R_A$). □

From Proposition 7.2, the results on parity games mentioned above, and Proposition 4.5 (for the lower bound) we immediately obtain:

Theorem 7.3. *The combined decision problem for CF-games with target DFA, extended context-free rules and left-to-right strategies is* EXPTIME*-complete.*

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 a game.*

8 Proofs for section 3

In this section we show how extended games can be transformed into (basically) equivalent pure games (Subsections 8.1 and 8.2). Further, the proofs for the normal forms are given in Subsection 8.3

The proof techniques are different for general and left-to-right strategies. In some cases we also need different proofs for different kinds of rule restrictions.

8.1 Transforming extended games for unrestricted strategies

We start this section by proving Lemma 3.1 which shows how to reduce an extended context-free game into a regular one. The proof is divided into three steps (Lemmas 8.1, 8.2 and 8.3), each step showing that the corresponding extension does not affect the expressive power of the game.

The first step shows how to remove navigation constraints.

Lemma 8.1. *For each CF-game $G = \langle \Sigma, R, T, N \rangle$ with navigation restrictions a CF-game $G' = \langle \Sigma', R', T' \rangle$ can be constructed in polynomial time such that, for each string w,* JULIET *wins (G, w) if and only if she wins (G', w). Furthermore, if G is unary, linear, or non-recursive then G' has the same properties. If G is d-bounded then G' is $3d + 1$-bounded.*

Proof. Let us first describe the idea behind the construction. Say, ROMEO chooses a rule $\alpha : A \to u$ with one navigation constraint $(\rightarrowtail, r, B)$, $B \in \Sigma$, in configuration (v, Aw). Hence,

in G, we get the configuration (v, uw) and, the next position choice of JULIET has to result in a configuration (vuw_1, aw_2) such that $w = w_1Bw_2$ and $w_1 \in L(r)$.

The corresponding sequence of configurations in the game G' should be as follows.

$$(v, Aw) \underset{(1)}{\to} (v, \boxed{\alpha} w) \underset{(2)}{\to} (v \boxed{\alpha} w_1, \hat{B}w_2) \underset{(3)}{\to} (v, uw_1\hat{B}w_2)$$

As can be seen here, we make use of new symbols. In G', there is one symbol $\hat{A}$ for each non-terminal A of R, and one symbol $\boxed{\alpha}$ for each rule α of R.

In step (1), ROMEO chooses the rule that is applied to A. In (2), JULIET chooses the position for the next move. Hence, w_1 has to match the regular expression r of the navigation constraint $(\rightarrowtail, r, B)$ of α. Finally, in (3), the rule chosen in (1) is applied, ending in a situation in which JULIET is obliged to choose the position of $\hat{B}$. The game could now continue with a step of kind (1) by using a rule with left-hand side $\hat{B}$. For navigation restrictions to the left the steps are corresponding.

Now we explain the construction of G' more formally. Let $G = \langle \Sigma, R, T, N \rangle$ be an extended game. The alphabet Σ' of G' consists of

$$\Sigma \cup \{\boxed{\alpha} \mid \alpha \text{ rule of } G\} \cup \{\hat{A} \mid A \text{ non-terminal of } G\}.$$

and of one *objection symbol* $B^\perp$, for each of the non-terminals B in this set. The intention is that ROMEO should be able to object if w_1, resulting from step (2), is not in $L(r)$.

For each non-terminal A of Σ, G' contains the rules $A \to A^\perp$ and $\hat{A} \to \hat{A}^\perp$. For rules $\alpha : A \to u$ in G without navigation constraints it contains the rules $A \to u$ and $\hat{A} \to u$. If α has navigation constraints there are the rules

$$A \to \boxed{\alpha},\ \hat{A} \to \boxed{\alpha} \text{ and } \boxed{\alpha} \to u.$$

If $\alpha : A \to u$ has no navigation constraint, then the last two rules are replaced by the rule $\hat{A} \to u$. It should be noted that the additional rules preserve all the properties mentioned in the statement of the lemma. Further, as each step is replaced by a sequence of at most three steps and because of the additional objection rules, G' is $3d + 1$-bounded if G is d-bounded.

The target language of G' has two purposes. First, it has to reflect the original target language of G. Second, it has to prevent ROMEO from abusing objection symbols. That is, all configurations in which ROMEO falsely selects an objection symbol become immediate winning configurations for JULIET. Note that T' will not accept any strings with more than one objection symbol. Consequently, after ROMEO places an objection symbol without getting into $L(T')$, he can choose such a symbol in each future step, preventing the string to ever enter $L(T')$. At the top level, T' is basically a disjunction of several automata. Besides one subautomaton T_0 which represents T, the other subautomata of T' correspond to the steps (1) to (3) above.

The subautomata of T' are described in the following.

- T_0 simply results from T by allowing each transition reading a symbol A also to read the corresponding $\hat{A}$.
- T_1 accepts all strings of the form $\Sigma^*(A^\perp + \hat{A}^\perp)\Sigma^*$, for each non-terminal A of G, to prevent ROMEO from taking an objection symbol if JULIET plays correctly during (1).
- T_2 ensures that ROMEO does not object unfairly in step (2). It accepts all strings of one of the following forms.

- $\Sigma^* \boxed{\alpha} r B^\perp \Sigma^*$, for each navigation constraint $(\rightarrowtail, r, B)$ of a rule $\alpha : A \to u$;
- $\Sigma^* \boxed{\alpha} r B^\perp \Sigma^*$, for each navigation constraint $(\rightarrowtail, r, *)$ of a rule $\alpha : A \to u$, and each $B \in \Sigma$;
- the corresponding forms for navigation constraints to the left.

– Finally, T_3 accepts strings of the form $\Sigma^* \boxed{\alpha}^\perp \Sigma^* \hat{A} \Sigma^*$ and $\Sigma^* \hat{A} \Sigma^* \boxed{\alpha}^\perp \Sigma^*$, for every rule α with navigation constraints and every $A \in \Sigma$.

Now we show that whenever one of the players departs from the intended behavior the other can win. We only consider the case of rules with navigation constraints. Note that a configuration containing more than one objection symbol is always winning for Romeo.

– In round (1), Juliet has to choose the position of a symbol of the form $\hat{A}$ if such a symbol is in the string. If she does not follow that rule, Romeo can object and win. If she does, Romeo looses if he objects (T_1).
– If there is no such symbol, she is free to choose an arbitrary position and Romeo looses if he objects (T_1).
– If Juliet chooses the position of $\boxed{\alpha}$ in round (2) again, Romeo can place a $\boxed{\alpha}^\perp$ and wins. If she chooses an allowed position which respects the navigation constraints, say labeled by B, then she wins in case Romeo selects $B^\perp$ (T_2).
– If, in round (3), Juliet does not go back to $\boxed{\alpha}$, then Romeo wins by choosing $\hat{B}^\perp$, in case she selected $\hat{B}$ or by some $A^\perp$, otherwise. If Romeo chooses $\boxed{\alpha}^\perp$ on the correct position then Juliet wins (T_3).

We only sketch how to deal with constraints for the first move of the play, given in $N(\epsilon)$. The idea is simply as follows. We add a disjoint copy Σ' of Σ to the alphabet of the game. We replace all symbols σ on the right-hand sides of rules by σ'. For each rule $A \to u$ we add a rule $A' \to u$. Finally, we modify T' accordingly, so that σ and σ' can be used interchangeably.

All symbols of the initial string are from Σ, after the first move, however, there is at least one non-Σ symbol. Hence, T' can be finally adapted to deal with wrong position choices of Juliet in the first move. □

The second step takes care of symmetric rules.

Lemma 8.2. *There are polynomial-time computable functions mapping CF-games G with symmetric rule choice and navigation constraints to CF-games G' with navigation constraints only, and pairs (G, w) to strings w' such that* Juliet *wins (G, w) if and only if* Juliet *wins (G', w'). If G is unary, linear or non-recursive then G' can be guaranteed to have the same property. If a non-recursive G is d-bounded then G' is $4d + 1$-bounded.*

Proof. Let G be the CF-game $\langle \Sigma, R, T, N, \Sigma_E \rangle$ with symmetric rule choice for non-terminals in Σ_E and navigation constraints N.

In the following we always suppose that A is a non-terminal from Σ_E and that $\alpha_1, \ldots, \alpha_k$ are all rules with left-hand side A in R, where $\alpha_i : A \to u_i$.

We describe two constructions, one which preserves unary and linear rules, and one which preserves bounded (but not necessarily unary) rules.

Preserving unary and linear rules. Let m be the maximum number of rules which are associated with a single symbol of G.

Let $w' = w\$^m$. The idea is as follows. Suppose that the current configuration of G is (u, Av), reflected by $(u, Av\$^m)$ in G'. If JULIET wants to choose rule $\alpha_i : A \to u_i$ then the play in G' should go through the following configurations.[7]

$$uAv\$^m \underset{(1)}{\to} u\hat{A}v\$^m \underset{(2)}{\to} u\hat{A}v\$^{i-1}\hat{\$}\$^{m-i}$$
$$\underset{(3)}{\to} u\boxed{\alpha_i}v\$^{i-1}\hat{\$}\$^{m-i} \underset{(4)}{\to} u\boxed{\alpha_i}v\$^m \underset{(5)}{\to} uu_iv\$^m$$

In (1), JULIET selects the position of A and ROMEO selects the only possible rule $A \to \hat{A}$. In (2), JULIET commits herself to the i-th rule by choosing the i-th \$ in the suffix of the string. This is rewritten by $\hat{\$}$. In (3), JULIET has to select the position of $\hat{A}$ and ROMEO will be forced to replace it by $\boxed{\alpha_i}$ to be consistent with the choice of the previously chosen \$. In step (4), JULIET selects the position of $\hat{\$}$ which is then rewritten by \$. Finally, in (5), JULIET selects $\boxed{\alpha_i}$ and ROMEO has to replace it by u_i. If the original rule $A \to u_i$ of G has a navigation constraint this is associated with the rule $\boxed{\alpha_i} \to u_i$. The other rules are associated with suitable navigation constraints that ensure moving back and forth between the \$ symbols and the symbols $\hat{A}$ and $\boxed{\alpha_i}$.

More precisely, the game G' is defined over the alphabet $\Sigma_1 = \Sigma' \cup \{\$, \hat{\$}\}$, where Σ' is defined as in Lemma 8.1.

The set R' contains the following rules. The numbers correspond to the 5 steps above.

(1) $A \to \hat{A}$ with navigation constraint $(\rightarrowtail, \Sigma_1^*, \$)$;
(2) $\$ \to \hat{\$}$ with all navigation constraints of the form $(\leftarrowtail, \Sigma_1^*, \hat{A})$, for $A \in \Sigma$;
(3) $\hat{A} \to \boxed{\alpha_i}$ with navigation constraint $(\rightarrowtail, \Sigma_1^*, \hat{\$})$, for each rule α_i of G;
(4) $\hat{\$} \to \$$ with all navigation constraints of the form $(\leftarrowtail, \Sigma_1^*, \boxed{\alpha_i})$, for rules α_i of G
(5) $\boxed{\alpha_i} \to u_i$, for each rule $\alpha_i : A \to u_i$ of G; if α_i had navigation constraints the same are associated with $\boxed{\alpha_i} \to u_i$.

Note that, due to the navigation constraints in G', JULIET never can select a position which is not allowed. Therefore there is no need to allow objection for ROMEO.

The automaton T' accepts all strings that correspond to strings accept to strings in $L(T)$, i.e., all strings of the form $u\$^m$ with $u \in T$. Furthermore, it accepts all strings resulting from a wrong rule choice of ROMEO in step (3). They are easily recognized by comparing the index i in $\boxed{\alpha_i}$ with the relative position of $\hat{\$}$ in the \$-suffix of the string.

Preserving non-recursive rules. The main idea in this case is as follows. We play the game on $w' = w$. The rules R' are the same as in R for non-terminals not in Σ_E.

If JULIET wants to choose rule $\alpha_i : A \to u_i$ in configuration (u, Av) then the play should go through the following configurations.

$$uAv \underset{(1)}{\to} u\langle\boxed{\alpha_1}\cdots\boxed{\alpha_k}\rangle v \underset{(2)}{\to} u\langle\boxed{\alpha_1}\cdots\boxed{\hat{\alpha}_i}\cdots\boxed{\alpha_k}\rangle v \underset{(3)}{\to} u\langle\boxed{\alpha_1}\cdots u_i\cdots\boxed{\alpha_k}\rangle v$$

Here $\langle$ and $\rangle$ are additional symbols. In step (1) JULIET selects the position of A and ROMEO has to replace it by $\boxed{\alpha_1}\cdots\boxed{\alpha_k}$. In step (2) chooses a rule α_i by selecting the corresponding symbol $\boxed{\alpha_i}$. ROMEO has to replace it by $\boxed{\hat{\alpha}_i}$. In step (3) JULIET has to select the position of

[7] We denote configurations only as strings as the chosen positions are always clear.

$\boxed{\hat{\alpha}_i}$ and ROMEO must rewrite it by u_i. The original navigation constraints of α_i are associated with the rule $\boxed{\hat{\alpha}_i} \to u_i$.

The rules of G' are as follows.

(1) Rules $A \to \langle \boxed{\alpha_1} \cdots \boxed{\alpha_k} \rangle$, for each non-terminal A of Σ with rules $\alpha_1, \ldots, \alpha_k$;
(2) Rules $\boxed{\alpha_i} \to \boxed{\hat{\alpha}_i}$, for each rule α_i of G;
(3) Rules $\boxed{\hat{\alpha}_i} \to u_i$, for each rule $\alpha_i : A \to u_i$ of G.

In order to prevent, in steps (2) and (3), JULIET from selecting a position outside $\langle \boxed{\alpha_1} \cdots \boxed{\alpha_k} \rangle$ or $\langle \boxed{\alpha_1} \cdots \boxed{\hat{\alpha}_i} \cdots \boxed{\alpha_k} \rangle$, there are objection symbols $\boxed{!}$ and rules $A \to \boxed{!}$ for every non-terminal of G'. T' accepts all strings in which an objection symbol occurs in a string $\langle \boxed{\alpha_1} \cdots \boxed{\alpha_k} \rangle$ or $\langle \boxed{\alpha_1} \cdots \boxed{\hat{\alpha}_i} \cdots \boxed{\alpha_k} \rangle$. Note that, after finishing a rule application properly, one symbol $\boxed{\alpha_i}$ has been replaced by a symbol from Σ, therefore there is at any time at most one substring of this form in the string.

Besides that T' accepts all strings u that result in strings of $L(T)$ after deleting all symbols not in Σ. □

It should be noted that the constructions given in the proof of Lemma 8.2 do *not* result in a game which is unary and non-recursive at the same time.

Finally we deal with games with several phases.

Lemma 8.3. *There are polynomial-time computable functions mapping extended CF-games G to CF-games G' with symmetric rule choice and navigation constraints only, and pairs (G, w) to strings w' such that* JULIET *wins (G, w) if and only if* JULIET *wins (G', w'). If G is unary, linear or non-recursive then G' can be guaranteed to have the same property. If a non-recursive G is d-bounded then G' is also $2d$-bounded.*

Proof. Let k be the number of phases in G, Σ its alphabet. For each symbol $A \in \Sigma$ we make use of the additional symbols $A_1, \ldots, A_k$.

Each move of G is simulated in G' by two moves. First, JULIET selects a position. Then, ROMEO replaces the symbol A of that position by A_i, where i is the number of the current phase. Then JULIET has to select the same position again and, depending on whether $A \in \Sigma_{E,i}$, JULIET or ROMEO chooses the rule which replaces A_i.

This process is guarded by the final symbols of the string. To this end, let $w' = w\$_1\$_2 \cdots \$_k$. In phase i the suffix of the string will be $\$'_1\$'_2 \cdots \$'_i\$_{i+1} \cdots \$_k$. We abbreviate the string $\$'_1\$'_2 \cdots \$'_i\$_{i+1} \cdots \$_k$ by D_i.

Let $\Sigma' = \Sigma \cup \Sigma_{\{1,\ldots,k\}} \cup \{\$_i, \$'_i \mid i \leq k\} \cup \{\bot\}$.

ROMEO can object for various reasons. JULIET could choose a wrong position in the second step (not A_i), or it could choose the $(i+1)$-st \$ before phase i is finished.

The rule set R' is defined as follows.

- For each non-terminal A and each $i \leq k$, G contains the rule $A \to A_i$.
- For each rule, $A \to u$ in R_i it has the rule $A_i \to u$.
- For each $i \leq k$ it contains the rule $\$_i \to \$'_i$.
- Additionally, it contains the rule $A \to \bot$, for each symbol $A \in \Sigma \cup \{\$_i, \$'_i \mid i \leq k\}$.

The new target language is defined as follows.

- For each i, it contains all strings of the form uD_i with $u \in L(T_i)$;

– For each i, j, $j \neq i$ and A, it contains all strings of the form $\Sigma^* A_j \Sigma^* D_i$, in order to prevent ROMEO choosing a symbol A_i not consistent with the current phase; It should be noted here, that at each time there is at most one symbol of the form A_i;
– For each $i < k$ it contains all strings of the form $u\$'_1\$'_2 \cdots \$'^{\perp}_i \$_{i+1} \cdots \$_k$ with $u \in W_i$; This is used to allow JULIET to go to phase $i+1$, if the current string is in W_i. If JULIET tries to replace the $(i+1)$-st \$ before the i-th phase is finished or to skip a phase then ROMEO wins.

As each rule application of G is simulated by two rule applications of G' and besides only depth 1 rules are added the depth of G' is at most $2d+1$ if the depth of G is d. □

Combining these three lemmas we eventually obtain Lemma 3.1.

8.2 Transforming extended games for left-to-right strategies

We now turn to the proof of Lemma 3.2. The proof is divided into two steps. The first step shows how to remove navigation constraints.

Lemma 8.4. *For each CF-game $G = \langle \Sigma, R, T, N \rangle$ with navigation restrictions a CF-game $G' = \langle \Sigma', R', T' \rangle$ can be constructed in polynomial time such that, for each string w,* JULIET *wins (G, w) with a left-to-right strategy if and only if* JULIET *wins (G', w) with a left-to-right strategy. Furthermore, if G being unary, linear and non-recursive is preserved. If G is d-bounded then G' is $(8d+1)$-bounded. If T is a DFA and each rule of R has at most one navigation restriction then T' is a DFA.*

Proof. The idea is similar to the proof of Lemma 8.1. Of course, we do not have to consider constraints that require navigation to the left.

For each rule $\alpha : A \to u$ with navigation constraints in R, R' contains the rule $A \to \boxed{\alpha}$. The other rules are as in R.

Whenever JULIET selects a wrong position wrt a constraint $(\rightarrowtail, r, B)$, ROMEO marks the selected symbol A by $A^{\perp}$. The target automaton T' has to ensure that, if ROMEO uses $A^{\perp}$ wrongly, the string between the previous position (the right-most symbol of the form $\boxed{\alpha}$) and $A^{\perp}$ matches the regular expression r. To this end, for each non-terminal A of G an objection rule $A \to A^{\perp}$ is added.

The target automaton T' accepts all strings v, for which $h(v) \in L(T)$, where h is the homomorphism with $h(A) = A$, for $A \in \Sigma$ and $h(\alpha) = u$, for $\alpha : A \to u$.

Furthermore, it accepts all strings that result from an objection of ROMEO in a situation where the current position choice of JULIET was valid with respect to the navigation constraints of the previous rule.

More precisely, for each rule $\alpha : A \to u$ with a navigation constraint $(\rightarrowtail, r, B)$ it accepts all strings of the form $u\boxed{\alpha}v_1 B^{\perp} v_2$, where $v_1 \in L(r)$. Similarly, for rules $(\rightarrowtail, r, *)$.

It is easy to see that T' can be made deterministic, if T is a DFA, each r is given by a DFA and there is at most one navigation constraint per rule. □

The second step allows to remove symmetric rules and therefore completes the proof of Lemma 3.2.

Lemma 8.5. *For each CF-game $G = \langle \Sigma, R, T, N, \Sigma_E \rangle$ with symmetric rule choice and navigation constraints a CF-game $G' = \langle \Sigma', R', T' \rangle$ with navigation constraints only can be constructed in polynomial time such that, for each string w,* Juliet *wins (G, w) with a left-to-right strategy if and only if* Juliet *wins (G', w) with a left-to-right strategy. Furthermore, the following statements hold.*

(a) If G is non-recursive then also G' is non-recursive. More precisely, if G is d-bounded then G' is $3d$-bounded.
(b) If G is unary (linear, resp.) and all Σ_E-rules have a single navigation restriction, then G' is also unary (linear, resp.).
(c) If T is a DFA and each rule of R has at most one navigation restriction then T' is a DFA.

Proof. For non-linear games the (second part of the) proof of Lemma 8.2 can be directly applied. Although it was stated in the case of unrestricted strategies, it works as well for left-to-right strategies.

Of course, we cannot take the same approach in the case of unary rules. Instead of letting Juliet choose a rule from a substring of rules we let her cycle through all possible rules. She selects the last rule by moving to the next position. It is necessary here, that the Σ_E-rules have navigation constraints because otherwise Juliet would be allowed to select the same position again.

To state it otherwise: if A is a Σ_E-non-terminal with rules $\alpha_1 : A \to u_1, \ldots, \alpha_m : A \to u_m$ and Juliet wants to select rule α_i then, for some v, w, the play proceeds as follows:

$$vAw \underset{(1)}{\to} v\boxed{\alpha_1}w \underset{(2)}{\to} v\boxed{\alpha_2}w \ldots \underset{(i)}{\to} v\boxed{\alpha_i}w,$$

and Juliet chooses a position inside w next. The corresponding rules are $A \to \boxed{\alpha_1}$ and, for each $j \leq m$, $\boxed{\alpha_{j-1}} \to \boxed{\alpha_j}$.

The automaton T' accepts all strings v for which $h(v) \in L(T)$, where $h(A) = A$, for each non-terminal A and $h(\boxed{\alpha}) = u$, for each rule $\alpha : A \to u$.

Both constructions result in a target DFA under the conditions of (c). □

8.3 Normal forms for games

We conclude this section by proving the normal form lemmas of Section 8.

Lemma 3.3. *For each CF-game $G = \langle \Sigma, R, T \rangle$ a CF-game $G' = \langle \Sigma', R', T' \rangle$ can be constructed in polynomial time such that T' is given by a DFA and for each string w,* Juliet *wins (G, w) if and only if* Juliet *wins $(G', w\$)$, where $\$$ is an additional symbol not occurring in Σ. Furthermore, if G is unary, linear or non-recursive then G' can be guaranteed to have the same property. If G is d-bounded then G' can be chosen $O(d)$-bounded.*

Proof. Let us first describe the idea of the construction: during a construction stage Juliet plays as in G. Eventually she obtains a word $w' = A_1 \cdots A_l \in L(T)$. In the transformation stage she replaces each symbol A of w' by a symbol $\boxed{A, q}$, where q is a state of T. For the resulting string $\boxed{A_1, q_1}, \ldots, \boxed{A_l, q_l}$ it should hold $q_i \in \delta(q_{i-1}, A_i)$, for every $i \leq l$, where q_0 is the initial state of T and $q_l \in F$.

How the transformation is done depends on the required properties of G' but is along the lines of Lemma 8.2.

If G' has to be linear (but not non-recursive) then, for each symbol A there are rules $A \to \boxed{A, q_1}$ and $\boxed{A, q_i} \to \boxed{A, q_{i+1}}$, for each $i < k$, where $q_1, \ldots, q_k$ are all states of T. Hence, Juliet can select the suitable symbol $\boxed{A, q_i}$ by cycling through all possible states.

If G' does not need to be linear then first a rule $A \to \langle \boxed{A, q_1} \cdots \boxed{A, q_k} \rangle$ is applied, and Juliet chooses a $\boxed{A, q_i}$ afterwards which is then replaced by q_i.

In both cases the automaton T' can be adapted to accept all strings obtained from w'. □

Note that in this proof we could not apply Lemma 3.1 as we often apply Lemma 3.3 only after application of Lemma 3.1 and Lemma 3.1 itself does not preserve determinism of games.

Lemma 3.4. *For each $d \geq 1$ there are polynomial-time computable functions transforming d-bounded CF-games G into 1-bounded CF-games G', and pairs (G, w) into strings w' such that* Juliet *wins (G, w) if and only if* Juliet *wins (G', w'). The same statement holds with respect to left-to-right strategies. Furthermore G' can be enforced to be unary or linear if G has the corresponding property and it is deterministic in case G is deterministic.*

Proof. We use the additional symbols $\$_0, \cdots, \$_{d-1}$ and $\#_1, \ldots, \#_d$. Let m be the maximal length of a right-hand side of a rule of G. The string w' is obtained by replacing each symbol A of w by $A\$_0 v_1^m$, where v_i, $i \leq d$, is recursively defined as follows.

- for $i < d$, v_i equals $\#_i \$_i v_{i+1}^m$.
- v_d is $\#_d$.

Roughly speaking, we attach to A the encoding of a tree of depth d and degree m, that will be filled stepwise by the string derived from A using rewriting rules. The symbols $\#_i$ are used to be replaced by symbols of Σ, the symbols $\$_i$ are used to store the rules that generated the symbols of Σ.

As an example (with $m = d = 2$), the G-configurations

$$AB \to CDB \to EDB \to EDF$$

will be simulated in G' by

$$\begin{aligned} A\$_0\#_1\$_1\#_2\#_2\#_1\$_1\#_2\#_2 B\$_0\#_1\$_1\#_2\#_2\#_1\$_1\#_2\#_2 \\ \to^* A\boxed{CD}C\$_1\#_2\#_2 D\$_1\#_2\#_2 B\$_0\#_1\$_1\#_2\#_2\#_1\$_1\#_2\#_2 \\ \to^* A\boxed{CD}C\boxed{E}E\#_2 D\$_1\#_2\#_2 B\$_0\#_1\$_1\#_2\#_2\#_1\$_1\#_2\#_2 \\ \to^* A\boxed{CD}C\boxed{E}E\#_2 D\$_1\#_2\#_2 B\boxed{F}F\$_1\#_2\#_2\#_1\$_1\#_2\#_2 \end{aligned}$$

The selection of a position in w and the replacement of a symbol A by a string $\alpha = A_1 \cdots A_n$ in the game G is mimicked in G' as follows. First, Juliet selects the position immediately to the right of the symbol A. This position carries a symbol of the form $\$_i$, where i is the depth of A in the current play. Then Romeo replaces $\$_i$ by the symbol $\boxed{\alpha}$. In the next n rounds the n symbols $\#_{i+1}$ attached to A must be replaced by $A_1, \ldots, A_n$. We thus have a rule $\$_i \to \boxed{\alpha}$ for each $i \leq d$ and each rule $\alpha \in R$, and a rule $\#_i \to a$ for each $i \leq d$ and each $a \in \Sigma$. Additionally, we have objection rules $\$_i \to \boxed{!!}$ and $\#_i \to \boxed{!}$. The target language of G' ensures that both players play according to this description:

– If JULIET selects a position which does not carry a $\$_i$ when considering a symbol of depth i then ROMEO can replace it by a symbol $\boxed{!!}$ and wins immediately.
– If JULIET does not select the positions labeled by $\#_{i+1}$ in the n copy moves, then ROMEO can place the symbol $\boxed{!}$ and wins.
– If ROMEO does not replace the j-th symbol labeled $\#_{i+1}$ by A_j then JULIET wins immediately.

All these conditions can be checked by a deterministic automaton (with d components) of polynomial size which works in parallel with the automaton of G (which simply ignores symbols of the form $\$_j$ and $\#_j$ and all symbols A, for which the right neighbor has been replaced by a symbol $\boxed{\alpha}$).

It should be stressed that the construction works for general strategies as well as for left-to-right strategies. □

9 Discussion

We have seen that in general it is undecidable to tell who wins a CF-game. We have also seen several restrictions on 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 largest string in $L(T)$. It is open whether this bound is tight. If ϵ-rules are allowed it is not even clear whether the game is decidable.

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 corresponds 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 thank Tova Milo who brought the problem to our attention, Serge Abiteboul and Victor Vianu for several fruitful discussions, and Tova Milo and Omar Benjelloun for the time they spent explaining us the beauty of AXML. We are greatly indebted to the referees of TOCS for their extremely careful reading and the numerous corrections they proposed.

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.*