Transducer-Based Rewriting Games for Active XML

Martin Schuster

TU Dortmund, Dortmund, Germany
martin.schuster@tu-dortmund.de

Abstract

Context-free games are two-player rewriting games that are played on nested strings representing XML documents with embedded function symbols. These games were introduced to model rewriting processes for intensional documents in the *Active XML* framework, where input documents are to be rewritten into a given target schema by calls to external services.

This paper studies the setting where dependencies between inputs and outputs of service calls are modelled by transducers, which has not been examined previously. It defines transducer models operating on nested words and studies their properties, as well as the computational complexity of the winning problem for transducer-based context-free games in several scenarios. While the complexity of this problem is quite high in most settings (ranging from NP-complete to undecidable), some tractable restrictions are also identified.

1998 ACM Subject Classification F.2.m Miscellaneous, F.4.2 Grammars and Other Rewriting Systems, H.3.5 Online Information Services

Keywords and phrases Active XML, Computational Complexity, Nested Words, Transducers, Rewriting Games

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.83

1 Introduction

Scientific context

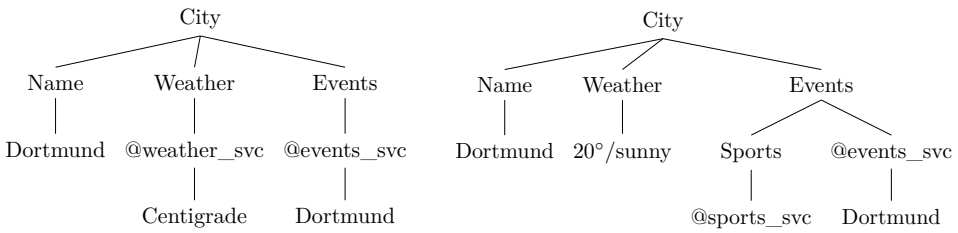
Context-free games on strings are two-player games extending context-free grammars, with the first player (called Juliet) choosing the non-terminal to be replaced and the second player (called Romeo) choosing a replacement for that non-terminal. The winning condition for Juliet is reaching, at some point during the game, some string in a given target language over the combined alphabet of non-terminals and terminals.

These games were first introduced in [7] to model the rewriting process of *Active XML (AXML)* [1] documents. The intention of AXML is modelling *intensional* documents, i.e. documents that do not store all required information explicitly but instead contain references to external services, from which current information may be materialised on demand, as illustrated in the example below. To this end, AXML extends standard XML with *function nodes* referring to external web services that may be called to insert data into the AXML document when the document is requested. Context-free games abstract from AXML to model the uncertainty inherent in using external data.

A standard example (cf. [6, 7]) of an application for AXML is depicted in Figure 1. In this example, we consider (part of) an AXML document retained by a local online news site providing information about current weather and events. Initially, the server-side document looks like the one in Figure 1a. The nodes labelled @weather_svc and @events_svc are function nodes referring to external weather and event services.

41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016).
Editors: Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier; Article No. 83; pp. 83:1–83:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



(a) Example document before rewriting. **(b)** Same document after function calls.

Figure 1 Example of Active XML rewriting. After calls to function nodes @weather_svc and @events_svc in Fig. 1a, external data is materialised to yield the document in Fig. 1b.

Figure 1b shows the document from Figure 1a after both function nodes have been called, replacing them by the external services' results. As exemplified by the function node labelled @weather_svc, call results replace the entire subtree rooted at the called function node, with that subtree being passed to the external service as a parameter. Concretely, the @weather_svc node's child tells the weather service that temperatures returned should be in centigrade. As the call result of the @event_svc node shows, returns of external services may contain further function nodes, even copies of the called function node.

The *safe rewriting problem* [6] of determining whether a given AXML document can always be rewritten into a target schema was abstracted in [7] into the problem of determining whether Juliet has a winning strategy in a given context-free game on strings, with Juliet representing a rewriting algorithm and Romeo representing the uncertainty inherent in function calls. This research assumed DTDs as schema formalisms. Allowing more expressive schema languages such as XML Schema [10] then led to research into context-free games on *nested strings* (i.e. XML-like linearisations of trees) [3]. Even though none of these previous works modelled dependencies between parameters and outputs of function calls, they already showed that the winning problem for Juliet can be undecidable or of a very high complexity, unless strategies for Juliet and allowed schema languages are seriously restricted.

The impact of service call parameters has so far only been studied in a limited fashion. In [10] (and in [6], for AXML rewriting with DTDs), external services were modelled by *input* (or *validation*) and *output* (or *replacement*) schemas, the semantics being that a function node could only be called if its parameter subtree was valid with regard to its corresponding input schema, which would then yield a return conforming to its output schema. This is a purely syntactic handling of input parameters which models a rather simple relationship between input and output of function calls; for instance, an @event_svc call reproducing its input parameters in its output as shown in Figure 1 cannot be enforced in this model.

While dependencies between parameters and outputs of service calls have always been implicit in the AXML model, they have not been studied in detail so far. Therefore, we extend the context-free games on nested words from [10] by (generally non-deterministic) transformations relating function parameters to possible outputs. We define *nested word transducers* (NWT) as a comparatively simple finite representation for transformations on nested words that naturally extends the nested word automata used in [10]. We then study the complexity of the winning problem for Juliet in various restrictions of context-free games with transducer-based replacement. As auxiliary results of potentially independent interest, we also examine closure properties and basic algorithmic problems of NWT.

Contributions

In light of prior undecidability and complexity results, the main objective of this paper is finding suitable restrictions to transducer-based context-free games that render the winning problem for Juliet decidable with as low complexity as possible. The two basic types of restrictions we examine are strategy restrictions (i.e. restrictions to Juliet's capabilities of calling function symbols) and restrictions to the type of NWT used for rewriting. To avoid undecidability, we only allow left-to-right strategies, i.e. once a function node has been called, no function calls to nodes preceding it (in post-order) are possible (cf. [7]).

The most important class of strategy restrictions considered here are restrictions to games with limited *replay*. In general context-free games, after a function call, Juliet continues her rewriting on that function call's result; we call this the *unbounded replay* case, as Juliet may continue rewriting function call results for as long as new function nodes are returned. In the *replay-free* case, Juliet is instead forbidden to call any function nodes inside results of function calls. As an intermediate case between unbounded replay and replay-free games, we also consider *bounded replay* games, where Juliet may only call functions returned by function calls up to a fixed maximum recursion depth. For instance, in a replay-free game, Juliet could call neither of the function nodes labelled @sports_svc and @events_svc in the situation of Figure 1b; in a bounded replay game of depth 2, on the other hand, she could call these nodes, but not any function nodes returned by those secondary calls.

The second type of restrictions comes from limiting expressiveness of the transducer used in games. Generally, transducers are allowed to be *non-functional*, i.e. any input string, may have several transducts (to model the fact that function call results are dependent upon, but not uniquely determined by, input parameters). The main types of transducers examined here are the following:

- *Nested word transducers (NWT)* allow for transforming input strings into output strings that are arbitrarily long, regardless of the input string's size.
- *Nested word transducers without ϵ-transitions (ϵ-free NWT)* may only increase the size of an input string by no more than a linear factor.
- *Relabelling transducers* may only change labels of input strings, not their structure.
- As a special case, *functional relabelling transducers* are relabelling transducers whose output string is uniquely determined by their input.

This paper's main complexity results are summarised in Table 1. The central insight here is that the least restricted settings yield an undecidable or non-elementary winning problem, and even for strong restrictions, the complexity of the winning problem is generally quite high, with no tractable case among the standard settings. For this reason, we also study several limitations of these settings, derived from our lower bound proofs, in order to reduce complexity:

- *Depth-bounded NWT* lower the complexity of the replay-free case to EXPSPACE-complete (in comparison to 2-EXPTIME for general NWT).
- Strategies with *bounded Call width* lower the complexity of the bounded-replay case for ϵ-free NWT from non-elementary to co-NEXPTIME-complete or co-NP-complete, depending on the precise formalisation of bounded Call width.
- *Write-once* strategies yield a tractable case for functional relabelling transducers in a setting even more restrictive than the replay-free one.

Table 1 Summary of complexity results. All results are completeness results.

	No replay	Bounded	Unbounded
NWT	2-EXPTIME	undecidable	undecidable
ϵ-free NWT	co-NEXPTIME	non-elementary	undecidable
Relabelling	PSPACE	PSPACE	EXPTIME
Functional relabelling	NP	NP	PSPACE

Related Work

Beyond the work already discussed, further complexity and decidability results for context-free games can be found in [2, 4], and [7] contains further references to related work.

Our concept of nested word transducers is based on Visibly Pushdown Transducers [8, 12], specifically the well-nested VPT of [5]. The original definitions of VPT in [8, 12] included ϵ-transitions, which were dropped from later definitions, as they caused several algorithmic problems, such as functionality and equivalence, to become undecidable (cf. [11]). Different from these approaches, this paper combines ϵ-transitions with the restriction to well-nested words, which is (to the best of the author's knowledge) new research.

Organisation

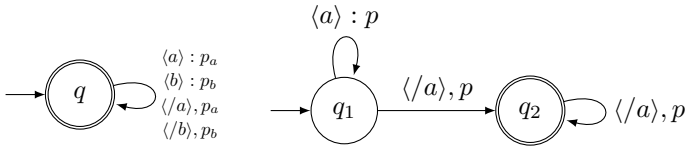
Section 2 gives basic notation and definitions. Section 3 defines nested word transducers and examines their structural and algorithmic properties. The next three sections give results on the complexity of the winning problem for games with transducer-based replacement, from most to least expressive – general nested word transducers (Section 4), nested word transducers without ϵ-transitions (Section 5), and relabelling transducers (Section 6). Each of these sections also discusses one of the restrictions with reduced complexity mentioned above. Section 7 concludes the paper. Due to space limitations, proofs and technical definitions are omitted here; for more details, an extended version is available online [9]. The author is grateful to Gaetano Geck and Thomas Schwentick for careful proof-reading and valuable suggestions, and to the anonymous reviewers for their insightful and constructive comments.

2 Preliminaries

For any natural number $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, \ldots, n\}$. For finite sets M, $\mathcal{P}(M)$ denotes the powerset of M, i.e. the set of all subsets of M. For an alphabet Σ, we denote the set of finite strings over Σ by Σ^* and ϵ denotes the empty string.

Nested words

For a finite alphabet Σ, $\langle\Sigma\rangle \stackrel{\text{def}}{=} \{\langle a \mid a \in \Sigma\}$ denotes the set of all *opening* Σ*-tags* and $\langle/\Sigma\rangle \stackrel{\text{def}}{=} \{\langle/a\rangle \mid a \in \Sigma\}$ the set of all *closing* Σ*-tags*. We denote by $\hat{\Sigma} \stackrel{\text{def}}{=} \langle\Sigma\rangle \cup \langle/\Sigma\rangle$ the set of all Σ-tags. The set $\mathrm{NW}(\Sigma) \subseteq \hat{\Sigma}^*$ of *(well-)nested words* (or *(well-)nested strings*) over Σ is the smallest set such that $\epsilon \in \mathrm{NW}(\Sigma)$, and if $u, v \in \mathrm{NW}(\Sigma)$ and $a \in \Sigma$, then also $u\langle a\rangle v\langle/a\rangle \in \mathrm{NW}(\Sigma)$. We (informally) associate with every nested word w its *canonical forest representation*, such that words $\langle a\rangle\langle/a\rangle$, $\langle a\rangle v\langle/a\rangle$ and uv correspond to an a-labelled leaf, a tree with root a (and subforest corresponding to v), and the forest of u followed by the forest of v, respectively. A nested string w is *rooted* if its corresponding forest is a tree. We denote the set of rooted nested strings over Σ by $\mathrm{rNW}(\Sigma)$. In a string $w = w_1 \ldots w_n \in \hat{\Sigma}^*$, two tags $w_i \in \langle\Sigma\rangle$ and $w_j \in \langle/\Sigma\rangle$ with $i < j$ are *associated* if the substring $w_i \ldots w_j$ of w is a rooted nested string. An opening (closing) tag w_i in w is *unmatched*, if it has no associated



(a) NWA A_1 **(b)** NWA A_2

Figure 2 NWAs A_1 and A_2 from Example 2.1.

closing (opening) tag in w. To stress the distinction from nested strings in $\mathrm{NW}(\Sigma)$, we refer to strings in Σ^* as *flat strings*.

Nested word automata

A *nested word automaton (NWA)* $A = (Q, P, \Sigma, \delta, q_0, F)$ [3] is basically a pushdown automaton which performs a push operation on every opening tag and a pop operation on every closing tag, and in which the pushdown symbols are just states. More formally, A consists of a set Q of *linear states*, a set P of *hierarchical states*, an alphabet Σ, a *transition relation* δ, an *initial state* $q_0 \in Q$, and a set $F \subseteq Q$ of *accepting (linear) states*. The relation δ is a subset of the union of sets $(Q \times \langle\Sigma\rangle \times Q \times P)$ and $(Q \times P \times \langle/\Sigma\rangle \times Q)$. We sometimes interpret δ as the union of two functions from $(Q \times \langle\Sigma\rangle)$ to $\mathcal{P}(Q \times P)$ and from $(Q \times P \times \langle/\Sigma\rangle)$ to $\mathcal{P}(Q)$ and write accordingly $(q', p) \in \delta(q, \langle a\rangle)$ for $(q, \langle a\rangle, q', p) \in \delta$ and $q' \in \delta(q, p, \langle/a\rangle)$ for $(q, p, \langle/a\rangle, q') \in \delta$. The semantics of NWA as well as the language $L(A)$ decided by a NWA A are defined in the natural way, with a NWA accepting if it reaches a configuration with an accepting state and empty stack. If A is a NWA, we call $L(A)$ a *regular language* (of nested words). A NWA is *deterministic* (or DNWA) if $|\delta(q, \langle a\rangle)| = 1 = |\delta(q, p, \langle/a\rangle)|$ for all $q \in Q$, $p \in P$ and $a \in \Sigma$. In this case, we simply write $\delta(q, \langle a\rangle) = (q', p')$ instead of $\delta(q, \langle a\rangle) = \{(q', p')\}$ (and accordingly for $\delta(q, p, \langle/a\rangle)$).

▶ **Example 2.1.** The NWA A_1 (Fig. 2a) checks that its input string is well-nested by pushing hierarchical state p_a (resp. p_b) to the stack on each opening $\langle a\rangle$ (resp. $\langle b\rangle$) tag and popping an according hierarchical state with each matching closing tag. In this manner, A_1 decides the set of all well-nested strings over $\{a, b\}$. The NWA A_2 (Fig. 2b) initially pushes a hierarchical state p each time it reads $\langle a\rangle$ in linear state q_1, then changes linear state to q_2 on reading the first $\langle/a\rangle$ and accepts iff each initial $\langle a\rangle$ is matched by a $\langle/a\rangle$. In this manner, it decides the language $\{\langle a\rangle^n \langle/a\rangle^n \mid n \geq 1\}$.

Context-free games

A *context-free game (with transduction) on nested words (cfG)* $G = (\Sigma, \Gamma, R, T)$ consists of a finite alphabet Σ, a set $\Gamma \subseteq \Sigma$ of *function symbols*, a *(replacement) rule set* $R \subseteq \mathrm{rNW}(\Sigma) \times \mathrm{NW}(\Sigma)$ and a *target language* $T \subseteq \mathrm{NW}(\Sigma)$. We will only consider the case where T is a non-empty regular nested word language and replacement rules are given by nested word transducers, to be defined in Section 3. A play of G is played by two players, Juliet and Romeo, on a word $w \in \mathrm{NW}(\Sigma)$. In a nutshell, Juliet moves the focus along w from left to right and decides for each closing tag $\langle/a\rangle$, whether she plays a *Read* or, in case $a \in \Gamma$, a *Call* move. In the latter case, Romeo then replaces the rooted word u ending at the position of $\langle/a\rangle$ with some word v with $(u, v) \in R$ and the focus is set on the first symbol of v. If no such word v exists, Romeo immediately wins the play. In case of a Read move, the focus just moves further on. Juliet wins a play if the word obtained at its end is in T.

Strategies

A *strategy* for player $p \in \{\mathrm{J}, \mathrm{R}\}$ maps game states where player p is to move into allowed moves for player p, i.e. strategies σ for Juliet return moves in $\{\mathrm{Read}, \mathrm{Call}\}$ while strategies τ for Romeo return replacement strings in $\mathrm{NW}(\Sigma)$. Given an initial word w and strategies σ, τ the play $\Pi(\sigma, \tau, w)$ according to σ and τ on w is uniquely determined. A *winning strategy* for Juliet is a strategy σ such that Juliet wins the play $\Pi(\sigma, \tau, w)$, for every τ of Romeo. By $\mathrm{JWin}(G)$ we denote the set of all words for which Juliet has a winning strategy in G.

The *Call depth* of a play Π is the maximum nesting depth of Call moves in Π, if this maximum exists. That is, the Call depth of a play is zero, if no Call is played at all, and one, if no Call is played inside a string yielded by a replacement move. For a strategy σ of Juliet and a string $w \in \mathrm{NW}(\Sigma)$, the *Call depth* $\mathrm{Depth}^G(\sigma, w)$ of σ on w is the maximum Call depth in any play $\Pi(\sigma, \tau, w)$. A strategy σ has *k-bounded Call depth* if $\mathrm{Depth}^G(\sigma, w) \leq k$ for all $w \in \mathrm{NW}(\Sigma)$. As a more intuitive formulation, we use the concept of *replay*: Strategies for Juliet of Call depth one are called *replay-free*, and strategies of k-bounded Call depth, for any k, have *bounded replay*.

Algorithmic problems

In this paper, we study the following algorithmic problem $\textsc{JWin}(\mathcal{G})$ for various classes $\mathcal{G}$ of context-free games with replacement transducers.

$\textsc{JWin}(\mathcal{G})$	
Given:	A context-free game $G \in \mathcal{G}$ and a string w.
Question:	Is $w \in \mathrm{JWin}(G)$?

A class $\mathcal{G}$ of context-free games in $\textsc{JWin}(\mathcal{G})$ generally comes with three parameters:

- the representation of the target language T,
- the representation of the replacement relation R, and
- to which extent replay is restricted.

We generally assume target languages to be represented by DNWAs, because the complexity of the winning problem is already quite high under that assumption, and our main interest is in finding classes $\mathcal{G}$ for which $\textsc{JWin}(\mathcal{G})$ is tractable. Replacement relations will be given as different types of nested word transducers (defined in Section 3). By a slight abuse of notation, the replacement transducer implementing a replacement relation R will also be referred to as R.

In each setting, we consider the cases of unrestricted replay, bounded replay (Call depth k, for some k), and no replay (Call depth 1). We note that replay depth is formally not an actual game parameter, but the algorithmic problem can be restricted to strategies of Juliet of the stated kind. If the class $\mathcal{G}$ of games is clear from the context, we often simply write JWin instead of $\textsc{JWin}(\mathcal{G})$.

3 Nested Word Transducers

In this section, we define nested word transducers and examine their closure properties and complexities of algorithmic problems. Thanks to our definition putting some rather severe restrictions on the use of ϵ-transitions and the allowed output of transducers, we obtain advantageous closure properties and comparatively low complexities.

Intuitively, a NWT T works much like a NWA with output and additional ϵ-transitions – T reads its input from left to right and decides nondeterministically which available transition

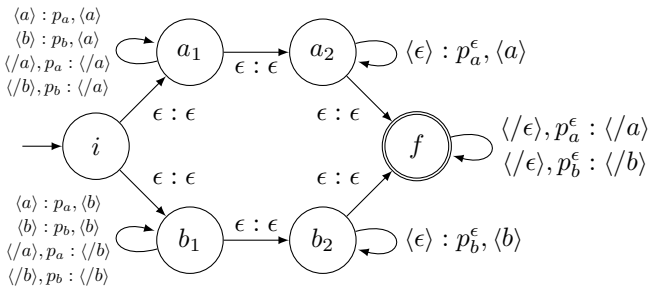


Figure 3 Nested Word Transducer T_{ab} from Example 3.2.

to use; on an opening (resp. closing) transition, it reads an opening (closing) input tag, changes its linear state and pushes (pops) a hierarchical state while producing an output. Opening (closing, internal) ϵ-transitions do not consume input symbols but induce state changes and outputs. T only produces an output string if it accepts the input string.

▶ **Definition 3.1.** A *nested word transducer* (or *NWT*) is a tuple $T = (Q, P, P_\epsilon, \Sigma, \delta, q_0, F)$ consisting of a set Q of *linear states*, a set P of *hierarchical states*, a set $P_\epsilon \subseteq P$ of *hierarchical ϵ-states*, an alphabet Σ, a *transition relation* δ, which is the union of three relations from $(Q \times (\langle\Sigma\rangle \cup \{\langle\epsilon\rangle\}) \times Q \times P \times \hat{\Sigma}^*)$ (called *opening* transitions), $(Q \times \{\epsilon\} \times Q \times \mathrm{NW}(\Sigma))$ (called *internal* transitions) and $(Q \times P \times (\langle/\Sigma\rangle \cup \{\langle/\epsilon\rangle\}) \times Q \times \hat{\Sigma}^*)$ (called *closing* transitions), an *initial state* $q_0 \in Q$, and a set of *accepting states* $F \subseteq Q$, such that for all $q, q', r, r' \in Q$, $p \in P$, $a \in \Sigma \cup \{\epsilon\}$ and $u, v \in \hat{\Sigma}^*$ it holds that[1]

- $(q, \langle\epsilon\rangle, q', p, u) \in \delta$ or $(q, p, \langle/\epsilon\rangle, q', u) \in \delta$ if and only if $p \in P_\epsilon$ (*ϵ-consistency*),
- if $(q, \langle a\rangle, q', p, u) \in \delta$ and $(r, p, \langle/a\rangle, r', v) \in \delta$, then $uv \in \mathrm{NW}(\Sigma)$ (*well-formedness*), and
- for each $(q, \langle a\rangle, q', p, u) \in \delta$ (resp. $(r, p, \langle/a\rangle, r', u) \in \delta$) with $u \neq \epsilon$, u contains at least one unmatched opening (resp. closing) tag (*synchronisation*).

As for standard NWA, we also write $(q', p, u) \in \delta(q, \langle a\rangle)$ (resp. $(q', u) \in \delta(q, p, \langle/a\rangle)$, $(q', u) \in \delta(q, \epsilon)$) instead of $(q, \langle a\rangle, q', p, u) \in \delta$ (resp. $(q, p, \langle/a\rangle, q', u), (q, \epsilon, q', u) \in \delta$).

A detailed semantics definition can be found in the extended version.

▶ **Example 3.2.** Figure 3 shows a NWT T_{ab}, with linear states displayed as circles and transitions as arrows. From the initial state i, T_{ab} branches nondeterministically into either state a_1 or b_1. In state a_1, T_{ab} checks that the input string is well-nested just as the NWA A_1 from Example 2.1. During this check, T_{ab} outputs $\langle a\rangle$ (resp. $\langle/a\rangle$) for each opening (resp. closing) input tag, effectively relabelling the input string to consist exclusively of a-labelled tags. In state a_2, T_{ab} inserts into the output string an arbitrary number of opening $\langle a\rangle$ tags, for which a matching number of $\langle/a\rangle$ tags are inserted in state f before T_{ab} accepts. The behaviour of T_{ab} in states b_1 and b_2 is analogous, but outputs consist only of b-labelled tags. Altogether, T_{ab} chooses nondeterministically some $x \in \{a, b\}$, relabels all tags of a well-nested input string into x-labelled tags and then appends a string of the form $\langle x\rangle^n \langle/x\rangle^n$.

The *image* $T(w)$ of a well-nested string $w \in \mathrm{NW}(\Sigma)$ under T is the set of all outputs of T on w according to some accepting run of T on w. This definition extends to sets of input strings in the natural way: For a set $S \subseteq \mathrm{NW}(\Sigma)$, we define $T(S) = \bigcup_{w \in S} T(w)$. The

[1] These three conditions make NWT roughly correspond to *synchronized visibly pushdown transducers* [8]; we mainly require them to ensure closure of regular nested word languages under NWT transduction.

domain $\mathcal{D}(T)$ of T is the set of all strings w such that $T(w) \neq \emptyset$, and the *range* $\mathcal{R}(T)$ of T is the set of all strings u such that there exists a $w \in \mathrm{NW}(\Sigma)$ with $u \in T(w)$, i.e. the set of all possible outputs of T.

We next define several restrictions on the expressiveness of NWT.

▶ **Definition 3.3.** Let $T = (Q, P, P_\epsilon, \Sigma, \delta, q_0, F)$ be a NWT. We call T

- ϵ*-free* if $P_\epsilon = \emptyset$ and δ contains no ϵ-transitions.
- *non-deleting* if the output component of every non-internal transition in δ is a non-empty string;
- *deterministic* (or a DNWT) if for every $q \in Q, p \in P$ and $a \in \Sigma$, it holds that $|\delta(q, \langle a\rangle)| = |\delta(q, p, \langle/a\rangle)| = 1$;
- a *relabelling* transducer if it is ϵ-free and for every $q, q' \in Q, p \in P$, $a \in \Sigma$ and $u \in \Sigma^*$, if $(q', p, u) \in \delta(q, \langle a\rangle)$, then $u \in \langle\Sigma\rangle$, and if $(q', u) \in \delta(q, p, \langle/a\rangle)$, then $u \in \langle/\Sigma\rangle$;
- *functional*, if for every $w \in \mathrm{NW}(\Sigma)$, it holds that $|T(w)| = 1$.

It is easy to see that the length of any output of an ϵ-free NWT is at most linear in the length of the input string, while outputs of general NWT may grow to an arbitrary length. We note that functionality, unlike the other restrictions defined here, is a *semantic* condition. We do not investigate in this paper the decidability or complexity of determining whether a NWT is functional; likely, techniques for Visibly Pushdown Transducers in [5] could be adapted for this purpose. Also, while determinism implies functionality, the converse does not hold.

The following lemma shows that we can assume without loss of generality that each transition of a NWT involves at most one input and at most one output symbol, i.e. each opening (closing) transition outputs at most one opening (closing) tag and each internal ϵ-transition outputs nothing.

▶ **Lemma 3.4.** *Each NWT* $T = (Q, P, P_\epsilon, \Sigma, \delta, q_0, F)$ *can be transformed in polynomial time into an NWT* $T' = (Q', P', P'_\epsilon, \Sigma, \delta', q_0, F)$ *with* $T(w) = T'(w)$ *for each* $w \in NW(\Sigma)$*, such that for any transition in* δ' *with output* u*, it holds that* $|u| \leq 1$*.*

We say that a NWT of this shape is in normal form*.*

In most of this paper, we restrict our attention to non-deleting transducers. This is because regular nested word languages are closed under transduction by non-deleting NWT, which does not hold in the presence of deletions (consider, for instance, a NWT deleting all matched opening and closing c-labelled tags on the regular input language $\{(\langle a\rangle\langle/a\rangle\langle c\rangle)^n(\langle/c\rangle\langle b\rangle\langle/b\rangle)^n \mid n \geq 0\}$). The practical motivation for desiring this property is the fact that the AXML setting assumes that function call results can be specified by standard XML schema languages, which are subclasses of regular nested word languages.

Moreover, for most of the transducer models examined here, non-deleting transducers are not a significant restriction when it comes to context-free games, as the following result states.

▶ **Lemma 3.5.** *Any context-free game* $G = (\Sigma, \Gamma, R, T)$ *with NWT* R *can be transformed in polynomial time into a game* $G' = (\Sigma', \Gamma, R', T')$ *such that* R' *is non-deleting and it holds that* $JWin(G') \cap NW(\Sigma) = JWin(G)$*.*

Using Lemma 3.4, it is comparatively easy (if tedious) to prove that non-deleting NWTs are closed under composition. This proof, like most proofs for properties of NWT in this section, follows proof ideas used in [5, 8] adapted to the specifics of NWT.

▶ **Proposition 3.6.** *Let T_1, T_2 be non-deleting NWT. Then there exists a non-deleting NWT T such that for all $w \in NW(\Sigma)$, it holds that $T(w) = (T_2 \circ T_1)(w) \stackrel{def}{=} T_2(T_1(w))$. This NWT T can be computed from T_1 and T_2 in polynomial time and is of size $\mathcal{O}(|T_1| \cdot |T_2|)$.*

Since we are solely interested in NWTs operating on well-nested strings, we restrict our attention to NWTs with well-nested domains. The following corollary to Proposition 3.6 justifies this restriction.

▶ **Corollary 3.7.** *Let T be a non-deleting NWT and A a NWA over alphabet Σ. Then, there exists a non-deleting NWT T' of size $\mathcal{O}(|T| \cdot |A|)$ such that $\mathcal{D}(T') = \mathcal{D}(T) \cap L(A)$ and $T'(w) = T(w)$ for each $w \in \mathcal{D}(T) \cap L(A)$.*

In order to prove closure of regular nested word languages under transduction by non-deleting NWT, we observe another helpful property of these transducers.

▶ **Lemma 3.8.** *Let T be a non-deleting NWT with $\mathcal{D}(T) \subseteq NW(\Sigma)$. Then $\mathcal{R}(T)$ is a regular language of nested words.*

▶ **Corollary 3.9.** *Regular nested word languages are closed under transduction by non-deleting NWT, i.e. if $L \subseteq NW(\Sigma)$ is regular and T an NWT, then $T(L)$ is regular.*

We now turn to the complexity of standard decision problems for NWT. The upper bounds use relatively simple constructions based on Proposition 3.6, while lower bounds follow from comparable results for NWA.

▶ **Theorem 3.10.** *The membership problem for non-deleting NWT (Given a non-deleting NWT T and strings $w, u \in NW(\Sigma)$, is $u \in T(w)$?) is in* PTIME.

▶ **Theorem 3.11.** *The nonemptiness problem for non-deleting NWT (Given a non-deleting NWT T, is there a string $w \in NW(\Sigma)$ with $T(w) \neq \emptyset$?) is* PTIME*-complete with regard to logspace reductions.*

▶ **Theorem 3.12.** *The type checking problem for non-deleting NWT (Given a non-deleting NWT T and NWA A_1, A_2, is $T(L(A_1)) \subseteq L(A_2)$?) is*
(a) EXPTIME*-complete in general, and*
(b) PTIME*-complete (w.r.t. logspace reductions) if A_2 is a DNWA.*

4 Games with general NWT replacement

Having laid the foundation with basic results on NWT, we now examine context-free games with NWT-based replacement. The main characteristic distinguishing general NWT from ϵ-free NWT is the fact that, for any input string w and NWT T, transducts in $T(w)$ may be arbitrarily large in the size of w. This behaviour is necessary if we want to simulate games with regular replacement languages (in the sense of [10]) by transducer-based games. As it turns out, however, NWT-based replacement is much more complex than that: the winning problem in games with replay becomes undecidable (as opposed to 2-EXPTIME with regular replacement languages), which may be proven by a rather straightforward reduction from the complement of the halting problem for Turing machines.

▶ **Theorem 4.1.** *For the class of games with NWT and Call depth $k \geq 2$,* JWin *is not recursively enumerable.*

Even the replay-free winning problem for JULIET is quite hard when using NWT for replacement – we show that this problem is complete for doubly exponential time. The lower bound uses a rather intricate reduction from a two-player tiling problem, while the upper bound is proven by reduction to the purely NWT-based problem of *alternating iterated transduction*, which can be proven to be in 2-EXPTIME.

▶ **Theorem 4.2.** *For the class of replay-free games with NWT,* JWIN *is* 2-EXPTIME-*complete.*

The lower bound proofs for both of these results require replacement transducers to output nested words of arbitrary depth. Considering our practical motivation, it is rarely required that function calls in Active XML documents return arbitrarily deep trees. Therefore, we now investigate the impact of limiting replacement transducers' output depth.

For simplicity's sake, we assume depth-boundedness as a *semantic* restriction, i.e. we assert that all outputs in $R(w)$ produced by a depth-bounded replacement transducer R on a string w obey a given upper bound on their depth, without examining the decidability and complexity of determining whether or not a given transducer is depth-bounded.

We note that NWT with an output depth linear in the size of the input string are already strictly more expressive than ϵ-free NWT, so the lower bounds from Section 5 also hold for NWT with linear output depth. As these lower bounds are already quite high, we focus only on transducers whose output depth is bounded by a constant.

▶ **Definition 4.3.** An NWT R is called *depth-bounded* if there is some constant $d \geq 0$ such that for any $w \in \mathrm{NW}(\Sigma)$ and any $w' \in R(w)$, the depth of w' is at most d.

Using depth-bounded NWT as replacement transducers places the complexity of the winning problem between those for general NWT and for ϵ-free NWT. The upper and lower bounds are proven similarly to those of Theorem 4.2, but use the fact that the stack size of a depth-bounded NWT on a fixed input is bounded by a constant.

▶ **Theorem 4.4.** *For the class of replay-free games with depth-bounded NWT,* JWIN *is* EXPSPACE-*complete.*

5 Games with ϵ-free NWT replacement

In this section, we examine context-free games with replacement relations given by ϵ-free NWT. As we shall see, this leads to a decidable winning problem for games with bounded replay, but non-elementary complexity in all but the easiest case. For the unbounded replay case, we can construct a rather straightforward reduction from the halting problem for TMs.

▶ **Theorem 5.1.** *For the class of games with ϵ-free NWT and unbounded replay,* JWIN *is undecidable.*

Different from games with general NWT, the winning problem for JULIET in games with ϵ-free NWT and fixed Call depth is decidable; however, the complexity of deciding JWIN is already non-elementary for Call depth 2.

▶ **Theorem 5.2.** *For the class of games with ϵ-free NWT and Call depth bounded by $d \geq 2$,* JWIN *is decidable, but not decidable in elementary time.*

Even for replay-free games with ϵ-free NWT, the complexity of deciding the winning problem for JULIET is still rather high. The lower bound is proven by reduction from a tiling problem, and the CO-NEXPTIME algorithm uses non-determinism to guess moves for ROMEO while trying out all possible strategies for JULIET by backtracking.

▶ **Theorem 5.3.** *For the class of replay-free games with ϵ-free NWT,* JWin *is complete for* co-NEXPTIME*.*

The non-elementary lower bound in Theorem 5.2 follows from the fact that, in each string returned by Romeo, Juliet may play Call arbitrarily often. On a return string corresponding to a path of length n, Juliet may play Call on all n nodes bottom-up, with each such Call doubling the number of nodes below the called node, inducing a non-elementary blow-up.

To avoid this, we now examine games with bounded *Call width*, where, intuitively, Juliet may only play Call for a bounded number of times in each replacement string given by Romeo. Note that Call width is counted within each individual replacement string – so, in a game of Call depth 3 and Call width c, if Juliet plays Call on some position of the input string, she may then place up to c calls within the string returned by Romeo, and again up to c calls in *each* of the depth-2 replacement strings resulting from those calls.

More formally, the Call width of a play Π is the maximum number of times Juliet plays Call in any replacement string given by Romeo in Π. This definition extends naturally into that of Call width of a strategy. Note that Call width only applies to *replacement* strings, so Juliet may still call arbitrarily many positions of the *input* string, even for games with Call width 0. For this reason, replay-free strategies always have bounded Call width.

The proof of Theorem 5.1 shows that JWin remains undecidable for games with unbounded Call depth, even with Call width bounded by 1. For bounded replay, though, the complexity of JWin collapses to that of the replay-free case if Call width is bounded.

▶ **Theorem 5.4.** *For the class of games with ϵ-free NWT, Call depth bounded by $d \geq 1$ and Call width bounded by $k \geq 1$,* JWin *is* co-NEXPTIME*-complete.*

As mentioned above, bounded Call width does not affect Juliet's options for Call moves on the input string, as we generally want Juliet to be able to at least process *all* function symbols in the input. Dropping this requirement (i.e. bounding *Call width including input*) yields at least an exponential improvement in complexity.

▶ **Theorem 5.5.** *For the class of games with ϵ-free NWT, Call depth bounded by d and Call width including input bounded by k,* JWin *is*

(a) co-NP*-complete for $d \geq 1$ and $k \geq 2$,*
(b) co-NP*-complete for $d \geq 2$ and $k \geq 1$, and*
(c) *in* PTIME *for $d = k = 1$.*

The upper bounds in Theorems 5.4 and 5.5 use a backtracking algorithm like the one for Theorem 5.3; in this case, however, bounded Call width reduces the size of both the decision tree for Juliet and the occurring replacement strings.

6 Games with relabelling replacement

As seen before, even the limited amount of insertion allowed by ϵ-free NWT renders the winning problem for Juliet quite complex. We now examine how this changes if we disallow insertion entirely. First, we show that the winning problem is greatly simplified by the fact that transducts of relabelling transducers do not require any additional space beyond that provided by the input. In fact, the upper bounds of Theorems 6.1 to 6.4 all use a nigh-trivial (alternating or nondeterministic) algorithm that simply simulates the game. Lower bounds, on the other hand, are proven by reduction from the word problem for linearly bounded (alternating) Turing machines (Theorems 6.1 and 6.3) and from standard logic-based problems (Theorems 6.2 and 6.4).

▶ **Theorem 6.1.** *For the class of games with relabelling transducers and unbounded replay,* JWin *is* EXPTIME*-complete.*

With limited or no replay, the complexity decreases even further.

▶ **Theorem 6.2.** *For any $k \geq 1$, for the class of games with relabelling transducers and bounded Call depth k,* JWin *is* PSPACE*-complete.*

As the winning problem for Juliet remains intractable (assuming PTIME $\neq$ PSPACE) for replay-free games with relabelling transducers, we now turn to the even more limited class of functional relabellings. Note that games with functional transducers are essentially "solitaire games" for Juliet, as they do not allow for any choice of transducts by Romeo.

▶ **Theorem 6.3.** *For the class of games with functional relabelling transducers and unbounded replay,* JWin *is* PSPACE*-complete.*

As for general relabelling transducers, the complexity of JWin is the same for games with bounded replay and no replay when restricted to functional relabelling transducers.

▶ **Theorem 6.4.** *For any $k \geq 1$, for the class of games with functional relabelling transducers and bounded Call depth k,* JWin *is* NP*-complete.*

We see that even in this very simple class of games, we still fail to obtain a PTIME upper bound. Careful examination of lower bound proofs shows that our semantics for replay-free games still allows for a sort of "hidden replay": On a string of the form $\langle a\rangle\langle b\rangle v\langle/b\rangle\langle/a\rangle$, if Juliet plays Call first on $\langle/b\rangle$ then on $\langle/a\rangle$, the substring v undergoes *two* transductions – one from the Call to $\langle/b\rangle$, another from the Call to $\langle/a\rangle$. This allows us to perform any number d of transductions on a given string by enclosing it inside d nested function symbols.

Excluding this hidden replay yields a very narrow restriction of context-free games, which we call *write-once* games. In these, no substring may be transduced more than once, i.e. Juliet may only play Call on any closing tag $\langle/a\rangle$ if the substring enclosed in it does not contain a substring on which Juliet has played Call before. Note that write-once games are always replay-free, but even weaker as far as Juliet's rewriting capabilities are concerned.

A slight adaptation of the proof of Theorem 6.2 shows that JWin remains PSPACE-hard for write-once games with arbitrary relabelling transducers; for functional (and deterministic) relabelling transducers, however, we can prove tractability. The proof constructs from a given game G a NWT R_{J} such that for each $w \in \mathrm{NW}(\Sigma)$, the set of all strings into which Juliet way rewrite w in G is given by $R_{\mathrm{J}}(w)$.

▶ **Theorem 6.5.** *For the class of write-once games with functional relabelling transducers,* JWin *is in* PTIME*.*

7 Conclusion

The research presented in this paper shows that a major challenge in using transducers for context-free games is finding sensible transducer models and strategy restrictions that do not cause a prohibitive increase in the complexity of the winning problem compared to context-free games without parameter transformation. This paper has made a first step towards identifying what suitable restrictions may look like; however, the few tractable cases identified here are still so restricted that they may be only of limited practical interest.

It is possible that the complexity of the winning problem in games with replacement transducers may be further reduced by restricting relevant schemas to be closer to practical

schema specifications for XML (such as DTDs or XML Schema). However, since research in [10] indicates that specifications of input schemas for external services influence the complexity of the safe rewriting problem, further research might be necessary to find transducer models whose input and output schemas can be described by DTDs or XML Schema.

References

1. Serge Abiteboul, Omar Benjelloun, and Tova Milo. The Active XML project: an overview. *VLDB J.*, 17(5):1019–1040, 2008. doi:10.1007/s00778-007-0049-y.
2. Serge Abiteboul, Tova Milo, and Omar Benjelloun. Regular rewriting of active XML and unambiguity. In *PODS*, pages 295–303, 2005. doi:10.1145/1065167.1065204.
3. Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3), 2009. doi:10.1145/1516512.1516518.
4. Henrik Björklund, Martin Schuster, Thomas Schwentick, and Joscha Kulbatzki. On optimum left-to-right strategies for active context-free games. In *ICDT*, pages 105–116, 2013. doi:10.1145/2448496.2448510.
5. Emmanuel Filiot, Jean-François Raskin, Pierre-Alain Reynier, Frédéric Servais, and Jean-Marc Talbot. Properties of visibly pushdown transducers. In *MFCS*, pages 355–367, 2010. doi:10.1007/978-3-642-15155-2_32.
6. Tova Milo, Serge Abiteboul, Bernd Amann, Omar Benjelloun, and Frederic Dang Ngoc. Exchanging intensional XML data. *ACM Trans. Database Syst.*, 30(1):1–40, 2005. doi:10.1145/1061318.1061319.
7. Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Active context-free games. *Theory Comput. Syst.*, 39(1):237–276, 2006. doi:10.1007/s00224-005-1278-3.
8. Jean-François Raskin and Frédéric Servais. Visibly pushdown transducers. In *ICALP (2)*, pages 386–397, 2008. doi:10.1007/978-3-540-70583-3_32.
9. Martin Schuster. Transducer-based rewriting games for Active XML. *CoRR*, abs/1606.02879, 2016. URL: http://arxiv.org/abs/1606.02879.
10. Martin Schuster and Thomas Schwentick. Games for Active XML revisited. In *ICDT*, pages 60–75, 2015. doi:10.4230/LIPIcs.ICDT.2015.60.
11. Frédéric Servais. *Visibly pushdown transducers*. Dissertation, ULB Belgique, 2011. URL: http://theses.ulb.ac.be/ETD-db/collection/available/ULBetd-09292011-142239/.
12. Alex Thomo, S. Venkatesh, and Ying Ying Ye. Visibly pushdown transducers for approximate validation of streaming xml. In *FoIKS*, pages 219–238, Berlin, Heidelberg, 2008. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=1786094.1786112.