The Membership Problem for the Shuffle of Two Deterministic Linear Context-Free Languages is NP-complete

Martin Berglund

Department of Computing Science, Umeå University
90187 Umeå, Sweden
mbe@cs.umu.se

Technical Report UMINF 12.09

Abstract. Formal language models which employ shuffling, or interleaving, of strings are of interest in many areas of computer science. Notable examples include system verification, plan recognition, and natural language processing. Membership problems for the shuffle of languages are especially interesting. It is known that deciding membership for shuffles of regular languages can be done in polynomial time, and that deciding (non-uniform) membership in the shuffle of two deterministic context-free languages is NP-complete. In this paper we narrow the gap by showing that the non-uniform membership problem for the shuffle of two deterministic *linear* context-free languages is NP-complete.

1 Introduction

In this paper we look at a membership problem for a language model based on the shuffle operator, $\odot$, introduced in [GS65]. This operator takes two strings, w and w', and returns the set of all possible interleavings of these strings. For example, $ab \odot cd = \{abcd, acbd, acdb, cabd, cadb, cdab\}$. We generalise the operator to sets in the usual way, $L \odot L' = \{w \odot w' \mid w \in L, w' \in L'\}$. The specific membership problem we consider is the non-uniform membership problem for the shuffle of two deterministic linear context-free languages. Specifically, we show that there exist deterministic linear context-free languages L and L' such that it is NP-complete to decide whether a given input string w is in the set $L \odot L'$, even if L and L' are fixed.

For listings of previous work see for example [BBH11,MRS98], additionally [HZ80] is of special interest, since it draws parallels between two-stack Turing machines and the shuffle of context-free languages. One important omission in [BBH11] (coauthored by the author of this paper) is [ORR78], which shows that the non-uniform membership problem for the shuffle of two deterministic context-free languages is NP-complete. In [BBH11] we show this for general context-free languages, unaware of [ORR78]. In this paper, however, that proof is extended further.

2 Preliminaries

Let $[n] = \{1, \ldots, n\}$ for all $n \in \mathbb{N}$. The cardinality of a set S is denoted $|S|$. An ordered set is a finite set S where the elements have a predetermined order. For $i \in [|S|]$ let $S(i)$ denote the ith element. When the set is stated with numbered elements $S = \{s_1, \ldots, s_n\}$ it is implied that $s_i = S(i)$.

The Kleene closure of a set S is denoted S^*. An alphabet is a finite set of symbols, usually denoted Σ. For strings $w, w' \in \Sigma^*$ let $w \cdot w'$ denote the concatenation, for sets of strings $W, W' \subseteq \Sigma^*$ let $W \cdot W' = \{w \cdot w' \mid w \in W, w' \in W'\}$. We may write the singleton set $\{w\}$ as w for simplicity.

Let $\alpha_1, \ldots \alpha_n \in \Sigma$ and $n \in \mathbb{N}$ (indices such as n being in $\mathbb{N}$ is usually left implicit going forward) in the following. Let ϵ denote the empty string. Let $w^{\mathcal{R}}$ denote the reverse of a string w, that is $(\alpha_1 \cdots \alpha_n)^{\mathcal{R}} = \alpha_n \alpha_{n-1} \cdots \alpha_1$. As usual let $|\alpha_1 \cdots \alpha_n| = n$, and for all $s \in \Sigma$ let $|\alpha_1 \cdots \alpha_n|_s = |\{i \in [n] \mid \alpha_i = s\}|$.

Deterministic linear context-free languages, denoted DLCF, will be used extensively in the following. It is assumed that the reader is familiar with the relevant formalisms for these languages (deterministic pushdown automata restricted to a single pushdown reversal for example), no formal definitions will be given here, see instead [HU90]. Instead of full pushdown automata implementations of the DLCF languages constructed (which would be large and hard to read) the strings in the languages are given in an inductive form from which the reader can easily construct automata themselves if desired.

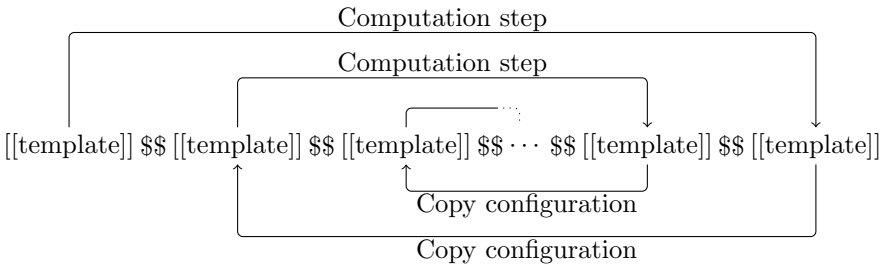
Non-deterministic polynomial time-bounded Turing machines are used heavily in the proofs to demonstrate NP-completeness. Full definitions of the machines are given, but for more complete background information on these topics see [GJ90,Min67].

3 Proof overview

The key building block necessary to make the shuffle of two DLCF languages perform a computation is making the languages communicate. This is done by constructing a template input string containing sequences of double-bracketed bits:

$$w = [[01]][[01]][[01]]\$\$[[01]][[01]][[01]]\$\$[[01]][[01]][[01]].$$

Assume that the *first* language contributes the string [0][1][0]$[1][1][1]$[1][0][1], then the *second* language *has to* contribute the string [1][0][1]$[0][0][0]$[0][1][0] if the whole input string w is to be assembled. Notice that the bit sequence in this string is the complement of the bit sequence in the first. In this way the two shuffled languages can communicate arbitrary choices by only accepting properly bracketed input. The proof will then use this to choose one language make computation steps for a Turing machine, while the other language copies the configuration around to link the computation up. The following figure acts as a visual aid to see how the languages will cooperate to simulate the computation (beware however that many details are left out, the figure only serves as a structural overview).



4 Parsing the Shuffle of Deterministic Linear Context-Free Languages

The reduction hinges on representing the computations of Turing machines as strings. To facilitate this we will make a somewhat specialised definition of non-deterministic Turing machine configurations and runs.

Definition 1. *A* non-deterministic Turing machine (NTM) *is a tuple* (Q, Δ) *where*

- Q *is the finite ordered set of states,*
- $\Delta : Q \times \{0,1\} \times \{\leftarrow, \rightarrow\} \times Q \times \{0,1\}$ *is the finite set of rules.*

$Q(1)$ *is the initial state,* $Q(|Q|)$ *is the accepting state.*

The following alphabet has all the symbols needed to complete the reduction.

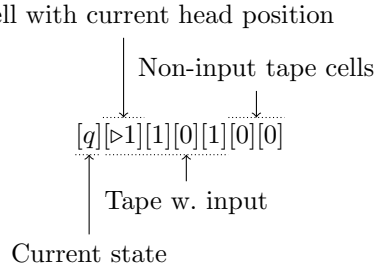
Definition 2. *Define* $\Sigma_M = Q \cup \Delta \cup \{0, 1, \triangleright, [,], \$, \#\}$.

A configuration becomes a simple string containing both the state, tape contents, and tape position, allowing rule applications to be expressed as string rewrites.

Definition 3. *The set of* configurations *of an NTM* $M = (Q, \Delta)$*, denoted* C_M*, is the set*

$$C_M = [\cdot Q \cdot] \cdot \{[0], [1]\}^* \cdot \{[\triangleright 0], [\triangleright 1]\} \cdot \{[0], [1]\}^* \subset \Sigma_M^*.$$

Example 1. As will be seen in Definition 5 an NTM will be provided with tape cells it can work on by padding the input with additional cells filled with zeros. For example an NTM with initial state q, the input string 1101, and 6 tape cells at its disposal would start in the following configuration.



Definition 4. *For an NTM $M = (Q, \Delta)$ we may* apply rule $r \in \Delta$ *to a configuration $c \in C_M$ to produce the configuration $c' \in C_M$ under the following conditions. Let $(q, \alpha, d, q', \alpha') = r$, then for all strings t_1 and t_2, and $\beta \in \{0, 1\}$*

- *if $d = \rightarrow$ and $c = [\cdot q \cdot] \cdot t_1 \cdot [\triangleright \cdot \alpha \cdot][\cdot \beta \cdot] \cdot t_2$ then $c' = [\cdot q' \cdot] \cdot t_1 \cdot [\cdot \alpha' \cdot][\triangleright \cdot \beta \cdot] \cdot t_2$,*
- *if $d = \leftarrow$ and $c = [\cdot q \cdot] \cdot t_1 \cdot [\cdot \beta \cdot][\triangleright \cdot \alpha \cdot] \cdot t_2$ then $c' = [\cdot q' \cdot] \cdot t_1 \cdot [\triangleright \cdot \beta \cdot][\cdot \alpha' \cdot] \cdot t_2$.*

We denote this rule application by $c \xrightarrow{r} c'$, or $c \rightarrow c'$ leaving r implicit.

Example 2. For example, in the configuration $[q][0][1][0][\triangleright 1][1][0]$ it is possible to apply the rule $(q, 1, \rightarrow, q', 0)$ to produce the configuration $[q'][0][1][0][0][\triangleright 1][0]$. In the configuration $[q][0][1][\triangleright 0]$ a rule $(q, 0, \rightarrow, q', 0)$ cannot be applied since there is no room to move to the right, nor can the rule $(q, 1, \leftarrow, q', 0)$, since $\triangleright$ is pointing to a 0 and the rule requires a 1.

Next follows the definitions of what it means for an NTM to accept a language in time bounded by some function. It should be obvious that this definition of a time-bounded non-deterministic Turing machine is equivalent to the usual one (see for example [Min67]). Most importantly this means that every problem $L \in$ NP (suitably encoded) is accepted by some NTM M in polynomially bounded time [GJ90].

Definition 5. *Take an NTM $M = (Q, \Delta)$, a function $\psi : \mathbb{N} \rightarrow \mathbb{N}$ and a string $\alpha_1 \cdots \alpha_n \in \{0, 1\}^*$. The* initial configuration *is defined as*

$$I(M, \psi, \alpha_1 \cdots \alpha_n) = [\cdot Q(1) \cdot] \underbrace{[\triangleright \cdot \alpha_1 \cdot] \cdots [\cdot \alpha_n \cdot][0][0] \cdots [0]}_{\psi(n)+1 \text{ bracketed bits}},$$

the set of final configurations *is $F(M) = ([\cdot Q(|Q|) \cdot] \cdot \Sigma_M^*) \cap C_M$.*

M accepts *$\alpha_1 \cdots \alpha_n$* in *ψ*-bounded time *if and only if the initial configuration can be transformed into some final configuration by* exactly *$\psi(n)$ rule applications. That is, there exists $\psi(n) + 1$ configurations, $c_1, \ldots, c_{\psi(n)+1}$ such that $c_1 = I(M, \psi, \alpha_1 \cdots \alpha_n)$, $c_{\psi(n)+1} \in F(M)$ and $c_i \rightarrow c_{i+1}$ for all $i \in [\psi(n)]$. The* language M accepts in *ψ*-bounded time *is exactly the set of strings M accepts in ψ-bounded time.*

This definition differs slightly from the usual one in that M is required to take *exactly* $\psi(n)$ steps to accept a string of length n, but any Turing machine that would accept the string in (the more usual) *at most* $\psi(n)$ steps can of course simply stay in the accepting state indefinitely to fulfil this condition.

The template string defined next will be used as the input for the membership query, encoding the Turing machine input and a long specially formatted suffix to make the shuffled computation possible.

Definition 6. *The* run template string *for running the machine $M = (Q, \Delta)$ in ψ-bounded time on the input string $\alpha_1 \cdots \alpha_n \in \{0, 1\}^*$ ($n \in \mathbb{N}$) is denoted $S(M, \psi, \alpha_1 \cdots \alpha_n)$ and is defined as follows. First the* configuration template *is*

$$T = [[\cdot Q(1) \cdots Q(|Q|) \cdot]] \underbrace{[[\triangleright 01]] \cdots [[\triangleright 01]]}_{\psi(n)+1 \text{ times}}.$$

Then $S(M, \psi, \alpha_1 \cdots \alpha_n)$ *equals*

$$I(M, \psi, \alpha_1 \cdots \alpha_n) \cdot \underbrace{\$\$ \cdot T \cdot \$\$ \cdot T \cdots \$\$ \cdot T}_{\psi(n) \text{ occurrences of } T} \cdot \$\$\#\# \cdot \underbrace{\$\$ \cdot T^{\mathcal{R}} \cdot \$\$ \cdot T^{\mathcal{R}} \cdot \$\$ \cdots T^{\mathcal{R}}}_{\psi(n)+1 \text{ occurrences of } T^{\mathcal{R}}}.$$

Example 3. Let $M = (\{q_1, q_2\}, \Delta)$, and let $\psi(2) = 1$, then $S(M, \psi, 1)$ is

$$[q_1][\triangleright 1][0]\$\$[[q_1 q_2]][[\triangleright 01]][[\triangleright 01]]\$\$\#\#\$\$]]10\triangleright[[\,]]10\triangleright[[\,]]q_2 q_1[[\$\$]]10\triangleright[[\,]]10\triangleright[[\,]]q_2 q_1[[.$$

The logical "bracketed" units are divided by a dotted line as a visual aid, since the $T^{\mathcal{R}}$ strings are made hard to read by their reversed brackets.

Next we define the concept of a shuffle complement with respect to a template.

Definition 7. *For all strings* $w, t \in \Sigma_M^*$*, let* $\mathit{complement}(w, t)$ *denote the* shuffle complement of w with respect to t*, defined as*

$$\mathit{complement}(w, t) = \{x \in \Sigma_M^* \mid t \in w \odot x\}.$$

Example 4. $\mathit{complement}([q_1][0][\triangleright 1], [[q_1 q_2 q_3]][[\triangleright 01]][[\triangleright 01]]) = \{[q_2 q_3][\triangleright 1][0]\}$.

A very small but important lemma follows.

Lemma 1. *For any configuration* $c \in C_M$ *and configuration template* T *(as in Definition 6) if it holds that* $|c|_[= \frac{1}{2}|T|_[$ *then*

1. $\mathit{complement}(c, T) = \{c'\}$ *for some string* c'*, and*
2. $\mathit{complement}(c', T) = \{c\}$.

Proof (sketch). If we have a configuration template string T as in Definition 6 and a configuration c, such that $|c|_[= \frac{1}{2}T_[$, then this means that T and c have the *same number of bracketed sections* (T has each section double-bracketed, $[[\triangleright 01]]$, c has each single-bracketed as in $[\triangleright 1]$). As a consequence $\mathit{complement}(c, T) = \{c'\}$ is a singleton. This is easy to see, by observing that the interleaving of c can only ever pick *one* of the [symbols in each [[pair in T, since it needs to read a] symbol before reading another left bracket. This forces it to skip the other bracket in the pair, meaning that the bracketed sections will match up one-to-one in the shuffle.

This in turn enforces that c' will *also* have $|c'|_[= \frac{1}{2}|T|_[$, and will have similarly single-bracketed sections, containing the complement of those in c with respect to the string $\triangleright 01$. The same argument therefore establishes that $\mathit{complement}(c', T) = \{c\}$. □

Next we define a deterministic linear context-free language which will encode the steps a given NTM can make.

Definition 8. *For an NTM* $M = (Q, \Delta)$ *the* step language for M*, denoted* $L_{step(M)}$*, is the smallest language that contains the string* #*, and all strings* $c_1 \cdot \$ \cdot l \cdot \$ \cdot c_2^{\mathcal{R}}$*, where* $l \in L_{step(M)}$*,* $c_1, c_2 \in C_M$*, and* $c_1 \xrightarrow{r} c_2$ *for some* $r \in \Delta$.

Example 5. Let $M = (\{q_1, q_2\}, \{(q_1, 0, \rightarrow, q_2, 1)\})$, then for example

$$[q_1][\triangleright 0][1][0]\$\#\$]0[\,]1\triangleright[\,]1[\,]q_2[\in L_{step(M)},$$

$$[q_1][0][\triangleright 0][0]\$\#\$]0\triangleright[\,]1[\,]1[\,]q_2[\in L_{step(M)},$$

$$[q_1][\triangleright 0][1][0]\$[q_1][\triangleright 0][1][0]\$\#\$]0[\,]1\triangleright[\,]1[\,]q_2[\$]0[\,]1\triangleright[\,]1[\,]q_2[\in L_{step(M)}.$$

It might not be immediately obvious that this language is both linear and deterministic, so let us look at how a deterministic linear push-down automaton can accept it. An automaton for $L_{step(M)}$ can start by pushing the first half of the string onto its stack, validating that it is in the regular language $(C_M \cdot \$)^*$ in the process. When it encounters # it switches to popping off the stack, while popping $c_1 \in C_M$ reading the reverse of $c_2 \in C_M$ on the string, and immediately rejecting unless c_2 differs from c_1 by exactly one rule application from Δ. The automaton can easily achieve this by checking that c_1 and c_2 are equal in all positions except the states and the immediate neighbourhoods of the $\triangleright$ symbol, both of which are constant-sized and can be remembered in the state of the automaton. It then simply validates that these differences correspond to a rule in Δ.

Now we turn to the other DLCF language, which is responsible for linking up the computation steps by making copies of the complement of configurations. It consists of strings of the form $\bar{c}_1 \cdot \$ \cdot \bar{c}_2 \cdots \bar{c}_2^{\mathcal{R}} \cdot \$ \cdot \bar{c}_1^{\mathcal{R}}$ where each $\bar{c}_i$ is such that $\{\bar{c}_i\} = \mathit{complement}(c, T)$ for some configuration c and configuration template T. Compare the constructed strings to those in Example 4.

Definition 9. *For an NTM $M = (Q, \Delta)$ the* inverted copy language for M, *denoted $L_{copy(M)}$, is defined as $L_{copy(M)} = \$ \cdot L$ where L is in turn defined as follows. First let*

- $\bar{Q}_i = [\,\cdot Q(1) \cdot Q(2) \cdots Q(i-1) \cdot Q(i+1) \cdots Q(|Q|) \cdot\,]$ *for* $i \in [|Q|]$,
- $U = \{[\triangleright 0], [\triangleright 1], [0], [1]\} \cdot \{[\triangleright 0], [\triangleright 1], [0], [1]\}^*$.

Then the strings in L are exactly the following. First, for all $t \in U$

$$\#\$ \cdot (\bar{Q}_{|Q|} \cdot t)^{\mathcal{R}} \in L.$$

Second, for all $\bar{c} \in \{\bar{Q}_i \mid i \in [|Q|]\} \cdot U$, and $l \in L_{copy(M)}$

$$\bar{c} \cdot \$ \cdot l \cdot \$ \cdot \bar{c}^{\mathcal{R}} \in L_{copy(M)}.$$

Example 6. Let $M = (\{q_1, q_2, q_3\}, \Delta)$, where q_3 is the final (last) state. Then among the strings in $L_{copy(M)}$ are

$$\#\$]0[\,]1\triangleright[\,]0\triangleright[\,]q_2q_1[,$$

$$[q_1q_3][\triangleright 0][\triangleright 1][1]\$\#\$]0[\,]1\triangleright[\,]0\triangleright[\,]q_2q_1[\$]1[\,]1\triangleright[\,]0\triangleright[\,]q_3q_1[,$$

$$[q_2q_3][1][\triangleright 0]\$[q_1q_3][\triangleright 0][\triangleright 1][1]\$\#\$]0[\,]1\triangleright[\,]0\triangleright[\,]q_2q_1[\$]1[\,]1\triangleright[\,]0\triangleright[\,]q_3q_1[\$]0\triangleright[\,]1\triangleright[\,]q_3q_2[.$$

It should be clear that this language is both deterministic and linear, the symbol # marking the centre playing a key role. The argument is similar to the one in the proof of Lemma 1, but slightly simpler, because no rules need to be taken into account.

This only leaves us to assemble the pieces to prove the main result.

Theorem 1. *Take any $w \in \{0,1\}^*$, NTM M and function $\psi : \mathbb{N} \to \mathbb{N}$. Then M accepts w in ψ-bounded time if and only if $S(M, \psi, w) \in L_{step(M)} \odot L_{copy(M)}$.*

This proof is divided into two lemmas, the first showing the "only if" direction, the second the "if" direction.

Lemma 2. *Take any string $\alpha_1 \cdots \alpha_n \in \{0,1\}^*$, NTM $M = (Q, \Delta)$ and function $\psi : \mathbb{N} \to \mathbb{N}$. If M accepts the string $\alpha_1 \cdots \alpha_n$ in ψ-bounded time then $S(M, \psi, \alpha_1 \cdots \alpha_n) \in L_{step(M)} \odot L_{copy(M)}$.*

Proof. Let $c_1, \ldots, c_{\psi(n)+1} \in C_M$ be the sequence of configurations which makes M accept $\alpha_1 \cdots \alpha_n$ (so $c_1 = I(M, \psi, \alpha_1 \cdots \alpha_n)$ and $c_{\psi(n)+1} \in F(M)$). Then construct the string

$$w_{step} = c_1 \cdot \$ \cdot c_2 \cdot \$ \cdots \$ \cdot c_{\psi(n)} \cdot \$\#\$ \cdot c^{\mathcal{R}}_{\psi(n)+1} \cdot \$ \cdot c^{\mathcal{R}}_{\psi(n)} \cdot \$ \cdots \$ \cdot c^{\mathcal{R}}_2.$$

Notice that $w_{step} \in L_{step(M)}$ by construction. Now, for each $i \in [\psi(n)+1]$ let $\{\bar{c}_i\} = \mathit{complement}(c_i, T)$ where T is a configuration template as in Definition 6. Recall that this complement is always a singleton. Now let

$$w_{copy} = \$ \cdot \bar{c}_2 \cdot \$ \cdot \bar{c}_3 \cdot \$ \cdots \bar{c}_{\psi(n)} \cdot \$\#\$ \cdot \bar{c}^{\mathcal{R}}_{\psi(n)+1} \cdot \$ \cdot \bar{c}^{\mathcal{R}}_{\psi(n)} \cdots \$ \cdot \bar{c}^{\mathcal{R}}_2.$$

It is then straightforward to check that $w_{copy} \in L_{copy(M)}$ by construction.

As an abbreviation denote the template string $S(M, \psi, \alpha_1 \cdots \alpha_n)$ by w. All that remains is to show that $w \in w_{step} \odot w_{copy}$. To illustrate:

$$\begin{aligned} w &= c_1 \$\$ T \$\$ \cdots \$ \;\; T \;\; \$\$\#\#\$\$ \;\; T^{\mathcal{R}} \;\; \$ \cdots \$ T^{\mathcal{R}}, \\ w_{step} &= c_1 \; \$ \; c_2 \; \$ \cdots \$ c_{\psi(n)} \;\; \$\#\$ \;\; c^{\mathcal{R}}_{\psi(n)+1} \$ \cdots \$ \, c^{\mathcal{R}}_2, \\ w_{copy} &= \quad \$ \; \bar{c}_2 \; \$ \cdots \$ \bar{c}_{\psi(n)} \;\; \$\#\$ \;\; \bar{c}^{\mathcal{R}}_{\psi(n)+1} \$ \cdots \$ \, \bar{c}^{\mathcal{R}}_2. \end{aligned}$$

w and w_{step} both start with c_1, so cancel that bit. Next w contains two dollar signs, one correponds to the initial in w_{copy} and one the next symbol in w_{step}. After that a T configuration template is next in w, c_2 is next in w_{step}, and $\bar{c}_2$ is next in w_{copy}. By construction $T \in c_2 \odot \bar{c}_2$, leaving us again with \$\$ next in w and a single \$ next in the other strings, and so on through all of w. □

Lemma 3. *Take any string $\alpha_1 \cdots \alpha_n \in \{0,1\}^*$, NTM $M = (Q, \Delta)$ and function $\psi : \mathbb{N} \to \mathbb{N}$. If $S(M, \psi, \alpha_1 \cdots \alpha_n) \in L_{step(M)} \odot L_{copy(M)}$ then M accepts $\alpha_1 \cdots \alpha_n$ in ψ-bounded time.*

Proof. Let $w = S(M, \psi, \alpha_1 \cdots \alpha_n)$, and take $w_{step} \in L_{step(M)}$ and $w_{copy} \in L_{copy(M)}$ such that $w \in w_{step} \odot w_{copy}$ (the lemma assumes these exist).

No string in $L_{step(M)} \cup L_{copy(M)}$ has two \$ symbols in a row, while every \$ occurrence in w consists of two \$ symbols. This enforces that every such \$\$ substring in w is divided up so that one belongs to w_{step} and one to w_{copy} (so $|w_{step}|_{\$} = |w_{copy}|_{\$} = \frac{1}{2}|w|_{\$}$). Combining this with the way $L_{step(M)}$ and $L_{copy(M)}$ are constructed it follows that the shuffling must have this structure

$$\begin{aligned} w &= c_1\$\$T\$\$\cdots\$\ T\ \$\$\#\#\$\$\ T^{\mathcal{R}}\ \$\cdots\$T^{\mathcal{R}}, \\ w_{step} &= c_1\ \$\ c_2\ \$\cdots\$c_{\psi(n)}\ \$\#\$\ d^{\mathcal{R}}_{\psi(n)+1}\$\cdots\$\,d^{\mathcal{R}}_2, \\ w_{copy} &= \ \ \$\ e_2\ \$\cdots\$e_{\psi(n)}\ \$\#\$\ e^{\mathcal{R}}_{\psi(n)+1}\$\cdots\$\,e^{\mathcal{R}}_2, \end{aligned}$$

for some configurations $c_1, \ldots, c_{\psi(n)}, d_2, \ldots, d_{\psi(n)+1} \in C_M$, and some strings $e_2, \ldots, e_{\psi(n)+1}$. That is, the assumption that $w \in w_{step} \odot w_{copy}$ does together with the placement of \$ symbols imply that

$$T \in c_i \odot e_i \quad \text{for all } i \in \{2, \ldots, \psi(n)\}, \tag{1}$$

$$T \in d_i \odot e_i \quad \text{for all } i \in \{2, \ldots, \psi(n)+1\}. \tag{2}$$

The second is not reversed since $T^{\mathcal{R}} \in d_i^{\mathcal{R}} \odot e_i^{\mathcal{R}} \iff T \in d_i \odot e_i$. Next, recall from Lemma 1 that *complement*(c_i, T) and *complement*(d_i, T) are singletons for all $i \in \{2, \ldots, \psi(n)\}$. Equations 1 and 2 dictate that $e_i \in$ *complement*(c_i, T) and $e_i \in$ *complement*(d_i, T), which means that *complement*$(c_i, T) =$ *complement*$(d_i, T) = \{e_i\}$. Reversing this (again by Lemma 1) yields *complement*$(e_i, T) = \{c_i\} = \{d_i\}$, so $c_i = d_i$. Let (the previously undefined) $c_{\psi(n)+1}$ be equal to $d_{\psi(n)+1}$ as well. The construction of $L_{step(M)}$ and $L_{copy(M)}$ dictates that

- $c_i \to d_{i+1}$, and therefore $c_i \to c_{i+1}$, for all $i \in [\psi(n)]$,
- $c_1 = I(M, \psi, \alpha_1 \cdots \alpha_n)$,
- *complement*$(e_{\psi(n)+1}, T) = \{c_{\psi(n)+1}\} \subset F(M)$ (since $e_{\psi(n)+1}$ does *not* contain the final state by construction).

From this it follows that $c_1, \ldots, c_{\psi(n)+1}$ is a correct configuration sequence which makes M accept $\alpha_1 \cdots \alpha_n$. □

Proof (of Theorem 1). Lemma 2 and Lemma 3 together show both directions of Theorem 1. □

It follows from Theorem 1 that the non-uniform membership problem for the shuffle of DLCF languages is NP-complete.

Corollary 1. *For an input string w it is an NP-complete problem to decide whether or not $w \in L \odot L'$ when L and L' are deterministic linear context-free languages, even when L and L' are fixed.*

Proof. The problem is trivially *in* NP. Membership in context-free languages can be decided in polynomial time, and we can, in polynomial time, guess any w_1 and w_2 such that $w = w_1 \odot w_2$ and check if $w_1 \in L$ and $w_2 \in L'$.

Hardness follows easily from Theorem 1. Pick any NTM M and *polynomial* function ψ such that M runs in ψ-bounded time. This characterises NP by

definition. Fix the languages $L = L_{step(M)}$ and $L' = L_{copy(M)}$. It is then possible to check if M would accept an input string w in ψ-bounded time by checking if $S(M, \psi, w) \in L \odot L'$. The reduction is polynomial since $S(M, \psi, w)$ produces a string that is of length $\mathcal{O}(\psi(|w|)^2)$ and can, because of its exceedingly simple structure, be constructed in time $\mathcal{O}(\psi(|w|)^2)$. Thus, choosing M such that it accepts an NP-complete language in polynomial time (e.g. a universal NTM) concludes the proof. □

5 Conclusions

Future work. The result in this paper narrows the gap between the cases where the membership problems for shuffled languages are intractable and where they are tractable. Still, there are several further restrictions that could be considered. The proof given here should be possible to modify in such a way that $L_{copy(M)}$ becomes a pure Dyck language, since the initial \$ symbol and the final configuration right after the \$#\$ midpoint marker are the only parts that disqualify it, but both of those could be handled by modifying the template string and changing $L_{step(M)}$. Similarly making both $L_{step(M)}$ and $L_{copy(M)}$ visibly pushdown [AM04] should be possible, since the structure of the construction is such that we know up-front which symbols will be pushed and which will be popped. Finding a language class larger than (or strictly different from) the regular languages for which membership in the shuffle is efficiently decidable remains an elusive but very interesting direction.

Acknowledgements. This paper would not have been possible without my shuffle collaborators Henrik and Johanna Björklund, and my advisor Frank Drewes.

References

[AM04] Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, STOC '04, pages 202–211, New York, NY, USA, 2004. ACM.

[BBH11] Martin Berglund, Henrik Björklund, and Johanna Högberg. Recognizing shuffled languages. In *Language and Automata Theory and Applications*, volume 6638 of *Lecture Notes in Computer Science*, pages 142–154. Springer Berlin / Heidelberg, 2011.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[GS65] Seymour Ginsburg and Edwin H. Spanier. Mappings of languages by two-tape devices. *J. ACM*, 12:423–434, July 1965.

[HU90] John E. Hopcroft and Jeffrey D. Ullman. *Introduction To Automata Theory, Languages, And Computation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1990.

[HZ80] David Haussler and H. Paul Zeiger. Very special languages and representations of recursively enumerable languages via computation histories. *Information and Control*, 47(3):201 – 212, 1980.

[Min67] Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

[MRS98] Alexandru Mateescu, Grzegorz Rozenberg, and Arto Salomaa. Shuffle on trajectories: Syntactic constraints. *Theoretical Computer Science*, 197(1–2):1–56, 1998.

[ORR78] William F. Ogden, William E. Riddle, and William C. Rounds. Complexity of expressions allowing concurrency. In *Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, POPL '78, pages 185–194, New York, NY, USA, 1978. ACM.