# Simulation Reductions for the Complexity of Subsequential Transducer Identification from Non-Characteristic Data

**William Schilling** WSCHILLI@ANDREW.CMU.EDU
*Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA*

## Abstract

This paper proposes a strategy for proving the computational hardness of finite-state transducer identification problems. Although it is well known that identifying classes of transductions characterized by state-counting bounds is NP-complete, the case of identifying structured classes of subsequential transductions is less understood, despite its relevance to current in-the-limit learning algorithms like SOSFIA. In this paper, we apply the proposed strategy to prove that both previous identification problems are NP-complete. This provides a simpler proof of the known result, as well as a new hardness bound on the identification of subsequential transducers when given a priori structural information in the sense of SOSFIA.

**Keywords:** transducer identification, structured classes of subsequential transductions, NP-completeness

## 1. Introduction

In the identification of transducers from samples of their behavior, the question of computational complexity is often subtle. A fundamental negative result is due to Gold (1978), who demonstrates that identification of Mealy-model transducers with at most $m$ states from given data is NP-complete. In contrast, positive results have been provided for well-behaved cases. The $L^*$ algorithm of Angluin (1987) identifies finite-state acceptors in polynomial time when given the ability to query data from a minimally adequate teacher. When the data is given passively instead of queried actively, the OSTIA algorithm of Oncina et al. (1993) identifies subsequential transducers in the limit—that is, as the data becomes sufficiently characteristic of the target transducer (Gold, 1967; de la Higuera, 2010)—in polynomial time.

Jardine et al. (2014) propose a further specialization to when the learner has access to the finite-state structure of the target transducer, making the learning target a *structured class of subsequential transductions*.[1] For this class, their SOSFIA algorithm provides identification in the limit with an improvement in time complexity compared to OSTIA. Applications of SOSFIA have been proposed in computational linguistics, where various sorts of phonological and morphological processes are modeled by structured classes of subsequential transductions.

Our work is motivated by a limitation of SOSFIA that is not shared by other in-the-limit learners like OSTIA. Whereas SOSFIA exactly identifies the target transducer when given

1. For consistency between the classes of $m$-state transducers studied by Gold (1978) and the name *structured classes of subsequential transductions*, we will call classes of subsequential transductions characterized by state-counting bounds *bounded classes of subsequential transductions*.

© 2026 W. Schilling.

sufficiently characteristic data, it may hypothesize a transducer that is inconsistent with the data when that data is insufficient. In other words, SOSFIA is not *feasible* in the sense of Gold (1978). This poses a challenge for linguistic applications, where it has been found that available data may fail to be fully characteristic of typical learning targets (Gildea and Jurafsky, 1996).

This paper establishes that SOSFIA's feasibility limitation is inherent, or that there exists no feasible learner for structured classes of subsequential transductions unless P = NP. The proof is by way of our proposed *simulation strategy*, where identification of an automaton $A$ is reduced to identification of an automaton $B$ by mapping data from $A$ to data that simulates $A$ within $B$. Specifically, by simulating stateless transducers within stateful transducers, we prove that the (feasible) identification of both bounded and structured classes of subsequential transducers is NP-complete. The hardness result for bounded classes is a subsequential transducer version of the result given by Gold (1978) for Mealy-model transducers, and uses a far simpler proof. The hardness result for structured classes is novel to our knowledge, and implies that SOSFIA's feasibility limitation is inherent.

The paper is organized as follows. Section 2 describes the theoretical preliminaries. Section 3 presents decision versions of the relevant identification problems, and known hardness results. Section 4 applies the simulation strategy to prove that identifying structured classes of subsequential transductions is NP-complete. Section 5 provides a reduction that results in a solver for identifying structured classes of subsequential transductions. Section 6 applies the simulation strategy to prove that identifying bounded classes of subsequential transductions is NP-complete. Section 7 concludes and proposes directions for future work.

## 2. Preliminaries

### 2.1. Strings and congruences

An alphabet $\Sigma$ is a finite nonempty set of characters. $\Sigma^*$ is the set of strings, or finite sequences of characters from $\Sigma$. $\lambda$ denotes the empty string. Let $u$, $v$, $w$, $x$ be strings. The length $|u|$ of $u$ is the number of characters in $u$. $uv$ or $u \cdot v$ is the concatenation of $u$ and $v$. If $w = uv$, then $u$ is a prefix of $w$. If $w = xuv$, then $u$ is a substring of $w$. A language over $\Sigma$ is a subset of $\Sigma^*$. A transduction between alphabets $\Sigma$ and $\Gamma$ is a maybe-partial function $t : \Sigma^* \to \Gamma^*$.

Let $\sim$, $\sim_1$, $\sim_2$ be equivalence relations over $\Sigma^*$. If $\sim$ results in finitely many equivalence classes, then it is said to have finite index. $\sim_1$ is finer than $\sim_2$ (equivalently, $\sim_2$ is coarser than $\sim_1$), denoted $\sim_1 \sqsubseteq \sim_2$, if and only if every equivalence class of $\sim_2$ is a union of equivalence classes of $\sim_1$. $\sim$ is a right congruence (or simply congruence) over $\Sigma$ if and only if for all strings $u$, $v$ and characters $\sigma$, we have $u \sim v \implies u\sigma \sim v\sigma$. A right congruence class (or simply congruence class) $\mathcal{C}$ over $\Sigma$ is a set of right congruences over $\Sigma$, all with finite index, that is closed under intersection and coarsening. If $\mathcal{C}$ contains a finest congruence $\sim$, then $\mathcal{C}$ is said to be structured. In this case, $\sim$ uniquely identifies $\mathcal{C}$, which consists of exactly the coarsenings of $\sim$, and is called the canonical congruence.

### 2.2. Finite automata and representations

A deterministic finite automaton (or simply automaton) over an alphabet $\Sigma$ is $\langle Q, \delta, q_0, F\rangle$ where $Q$ is a finite set of states, $\delta : Q \times \Sigma \to Q$ is the maybe-partial transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. The extended transition function $\delta^* : Q \times \Sigma^* \to Q$ is defined via $\delta^*(q, uv) = \delta^*(\delta^*(q, u), v)$, $\delta^*(q, \sigma) = \delta(q, \sigma)$, and $\delta^*(q, \lambda) = q$ for all $q \in Q$, $u, v \in \Sigma^*$, and $\sigma \in \Sigma$. The domain of $\delta^*$ is restricted according to the domain of $\delta$. Every automaton $A$ recognizes a language $\mathcal{L}(A)$ defined as $\{u \in \Sigma^* \mid \delta^*(q_0, u) \in F\}$. An automaton is total if and only if $\delta$ is total and $F = Q$. In this case, $\delta^*$ is also total and $\mathcal{L}(A) = \Sigma^*$.

Strings $u$, $v$ are transition congruent with respect to $A$, denoted $u \approx_A v$, if and only if $(q_0, u) \notin \mathrm{dom}(\delta^*) \wedge (q_0, v) \notin \mathrm{dom}(\delta^*)$ or $\delta^*(q_0, u) = \delta^*(q_0, v)$. The transition congruence is a right congruence. Conversely, every right congruence $\sim$ is the transition congruence of the total finite automaton $A_\sim$ in Equation 1. This correspondence enables us to take total finite automata as a representation of congruences and structured congruence classes, which is analogous to SOSFIA's use of output-empty transducers to specify the transition structures of learning targets (Jardine et al., 2014).

$$\begin{aligned} A_\sim &= \langle \Sigma^*/\sim, \delta_\sim, [\lambda]_\sim, \Sigma^*/\sim\rangle \\ \delta_\sim([u]_\sim, \sigma) &= [u\sigma]_\sim \end{aligned} \tag{1}$$

We will also want access to a notion of congruence class that is not predicated on the specification of a domain $\Sigma$ (e.g. Definition 5). To this end, a structured congruence universe $\mathfrak{C}$ is defined as a polynomial-time computable function that associates each alphabet $\Sigma$ with a structured congruence class $\mathfrak{C}_\Sigma$ over $\Sigma$.

### 2.3. Transducers and subsequential transductions

A transduction $t : \Sigma^* \to \Gamma^*$ is homomorphic if and only if for all strings $u$, $v$, we have $t(uv) = t(u)t(v)$. A homomorphic transducer is a maybe-partial function $h : \Sigma \to \Gamma^*$. Every homomorphic transducer recognizes a homomorphic transduction $t : \Sigma^* \to \Gamma^*$ defined via $t(uv) = t(u)t(v)$, $t(\sigma) = h(\sigma)$, and $t(\lambda) = \lambda$ for all $u, v \in \Sigma^*$ and $\sigma \in \Sigma$, where the domain of $t$ is restricted according to the domain of $h$. Conversely, every homomorphic transduction is recognized by the homomorphic transducer defined via $h(\sigma) = t(\sigma)$.

A subsequential transducer $T$ between alphabets $\Sigma$ and $\Gamma$ is $\langle A, i, o, f\rangle$ where $A = \langle Q, \delta, q_0, F\rangle$ is the underlying automaton, $i \in \Gamma^*$ is the initial output string, $o : \mathrm{dom}(\delta) \to \Gamma^*$ is the transition output function, and $f : F \to \Gamma^*$ is the final output function. The extended transition output function $o^* : Q \times \Sigma^* \to \Gamma^*$ is defined via $o^*(q, uv) = o^*(q, u) \cdot o^*(\delta^*(q, u), v)$, $o^*(q, \sigma) = o(q, \sigma)$, and $o^*(q, \lambda) = \lambda$ for all $q \in Q$, $u, v \in \Sigma^*$, and $\sigma \in \Sigma$. The domain of $o^*$ is restricted according to the domain of $o$. Every subsequential transducer recognizes a transduction $t : \Sigma^* \to \Gamma^*$ defined via $t(u) = i \cdot o^*(q_0, u) \cdot f(\delta^*(q_0, u))$ with $\mathrm{dom}(t) = \mathcal{L}(A)$. A transduction is called subsequential if and only if it is recognized by some subsequential transducer.

Let $T = \langle A, i, o, f\rangle$ be a subsequential transducer that recognizes the subsequential transduction $t$. For all $m \in \mathbb{N}^+$, if $A$ has at most $m$ states, then $T$ and $t$ are called $m$-state. For all congruence classes $\mathcal{C}$, if $\approx_A$ is in $\mathcal{C}$, then $T$ and $t$ are called $\mathcal{C}$-subsequential. For all $m \in \mathbb{N}^+$ and structured congruence classes $\mathcal{C}$, the $m$-state subsequential transductions are

a *bounded class of subsequential transductions*, and the $\mathcal{C}$-subsequential transductions are a *structured class of subsequential transductions*.

### 2.4. Computational complexity

We will assume that the reader is familiar with the NP complexity class, NP-completeness, and polynomial-time mapping reductions. For an introduction, the reader is directed to Sipser (2013).

## 3. Transducer Identification Problems

Definition 1 is the Homomorphic Transducer Identification Problem (HTIP). Ehrenfeucht and Rozenberg (1979) prove that the version of the problem with instances restricted to just one data pair is NP-complete. Since those problem instances are a special case of HTIP instances, HTIP is also NP-hard, a fact used in Lemma 2.

**Definition 1** *For a fixed alphabet* $\Gamma$*, the* Homomorphic Transducer Identification Problem*, denoted* $\Gamma$-HTIP*, is as follows: given an alphabet* $\Sigma$ *and* $n$ *data pairs* $(x_1, y_1), \ldots (x_n, y_n) \in \Sigma^* \times \Gamma^*$*, decide whether there exists a homomorphic transduction* $t : \Sigma^* \to \Gamma^*$ *such that* $t(x_i) = y_i$ *for all* $i$*.*

**Lemma 2** $\Gamma$-HTIP *is* NP*-complete for all alphabets* $\Gamma$ *where* $|\Gamma| \geq 2$*.*

**Proof** Firstly, $\Gamma$-HTIP is in NP for all alphabets $\Gamma$. The desired verifier takes a homomorphic transducer as a witness and checks that the homomorphic transduction $t$ it recognizes is such that $t(x_i) = y_i$ for all $i$. It remains to show that $\Gamma$-HTIP is NP-hard for all alphabets $\Gamma$ where $|\Gamma| \geq 2$, for which we appeal to Ehrenfeucht and Rozenberg (1979) ■

Definition 3 is the Bounded Subsequential Transducer Identification Problem (BSTIP), the problem of identifying bounded classes of subsequential transductions. Gold (1978) proves that a similar problem, with transducers restricted to the Mealy model, is NP-complete. It is easy to show that BSTIP is in NP, as in Lemma 4.

**Definition 3** *For a fixed alphabet* $\Gamma$ *and* $m \in \mathbb{N}^+$*, the* Bounded Subsequential Transducer Identification Problem*, denoted* $(\Gamma, m)$-BSTIP*, is as follows: given an alphabet* $\Sigma$ *and* $n$ *data pairs* $(x_1, y_1), \ldots (x_n, y_n) \in \Sigma^* \times \Gamma^*$*, decide whether there exists an* $m$*-state subsequential transduction* $t : \Sigma^* \to \Gamma^*$ *such that* $t(x_i) = y_i$ *for all* $i$*.*

**Lemma 4** $(\Gamma, m)$-BSTIP $\in$ NP *for all alphabets* $\Gamma$ *and* $m \in \mathbb{N}^+$*.*

**Proof** The desired verifier takes an $m$-state subsequential transducer as a witness and checks that the $m$-state subsequential transduction $t$ it recognizes is such that $t(x_i) = y_i$ for all $i$. ■

Definition 5 is the Structured Subsequential Transducer Identification Problem (SSTIP), the problem of identifying structured classes of subsequential transductions. It is easy to show that SSTIP is also in NP, as in Lemma 6.

**Definition 5** *For a fixed alphabet* $\Gamma$ *and structured congruence universe* $\mathfrak{C}$*, the* Structured Subsequential Transducer Identification Problem*, denoted* $(\Gamma, \mathfrak{C})$-SSTIP*, is as follows: given an alphabet* $\Sigma$ *and* $n$ *data pairs* $(x_1, y_1), \ldots (x_n, y_n) \in \Sigma^* \times \Gamma^*$*, decide whether there exists a* $\mathfrak{C}_\Sigma$*-subsequential transduction* $t : \Sigma^* \to \Gamma^*$ *such that* $t(x_i) = y_i$ *for all* $i$*.*

**Lemma 6** $(\Gamma, \mathfrak{C})$-SSTIP $\in$ NP *for all alphabets* $\Gamma$ *and structured congruence universes* $\mathfrak{C}$*.*

**Proof** The desired verifier takes a $\mathfrak{C}_\Sigma$-subsequential transducer as a witness and checks that the $\mathfrak{C}_\Sigma$-subsequential transduction $t$ it recognizes is such that $t(x_i) = y_i$ for all $i$. ■

An observation that will be useful in our proofs is that for every class of transductions in these identification problems, if there is a transduction from the class that is consistent with the given data, then there is also a *total* such transduction, since its domain can be extended without changing its original outputs. This allows us to assume without loss of generality that the witness for a problem instance contained in any of these identification problems is total.

## 4. Identification of Structured Classes of Subsequential Transductions

This section applies the simulation strategy to reduce HTIP to SSTIP, proving that SSTIP is NP-complete. The reduction relies on the fact that the maximally-refined transition structure of a $\mathcal{C}$-subsequential transducer is known a priori to be the canonical congruence of $\mathcal{C}$. It works by isolating one state in this canonical transition structure around which to simulate a single-state machine, that is, a homomorphic transducer. Treating the transition structure as a directed graph with its states as vertices and its transitions as labeled edges, we will call a state that can be isolated in this way a *trackable vertex*, as in Definition 7.

**Definition 7** *Let* $(V, E)$ *be a directed graph with labeled edges* $(v, \sigma, v')$ *where* $v, v' \in V$*. For all* $v_0, v \in V$*,* $v$ *is* trackable *from* $v_0$ *if and only if there exists a* tracking path*, a path from* $v_0$ *to* $v$ *that contains every outgoing neighbor of* $v$*.*

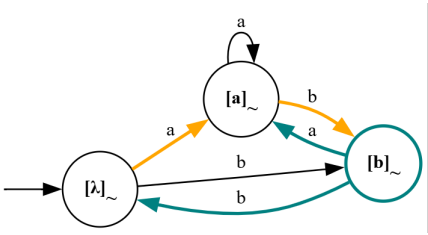
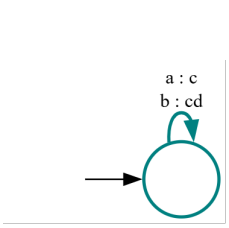


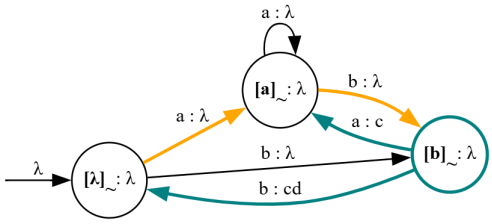

Figure 1: $[b]_\sim$ is a trackable vertex from $[\lambda]_\sim$, and the string $ab$ traverses its tracking path.

A vertex $v$ being trackable guarantees that we can traverse from the initial vertex $v_0$ to $v$, and that we can always traverse back to $v$ after traversing away from $v$. This is easy to see in Figure 1. Now, a homomorphic transducer processing an input string $x$ can be simulated on the outgoing transitions of $v$ by inserting the string that traverses the tracking

path back to $v$ before every character in $x$. This way, it is exactly the original characters in $x$ that traverse the outgoing transitions of $v$. To ensure that the inserted characters cannot produce additional output, it suffices to enforce that the input string traversing the entire tracking path maps to the output string $\lambda$. This way, every output string along the tracking path must be $\lambda$. Figure 2 visualizes this reduction outline with the transition structure from Figure 1.




$\texttt{data} = \{(a, c),$
$(ab, ccd),$
$(ba, cdc)\}$




$f_\sim(\texttt{data}) = \{(ab \cdot ab, c),$
$(ab \cdot ab \cdot bab, c \cdot cd),$
$(ab \cdot bab \cdot ab, cd \cdot c),$
$(ab, \lambda)\}$

(a) An HTIP instance and a witness (b) The mapped SSTIP instance and witness

Figure 2: The HTIP witness is simulated around the $[b]_\sim$ state in the SSTIP witness.

Algorithm 1 shows that the desired trackable vertex always exists by outputting such a vertex and a corresponding tracking path. The algorithm is essentially a depth-first search without backtracking, since a terminal node in the search is exactly a trackable vertex. Lemma 8 proves correctness.

**Algorithm 1:** finding a trackable vertex and a corresponding tracking path

```
1  Function find_trackable_vertex(V, E, v_0):
2      mark(v_0);
3      p ← singleton_path(v_0);
4      v ← v_0;
5      loop
6          if ∃(v, σ, v') ∈ E, ¬is_marked(v') then
7              mark(v');
8              p ← append_to_path(p, (v, σ, v'));
9              v ← v';
10         else
11             return (v, p);
```

**Lemma 8** *Let $(V, E)$ be a directed graph with labeled edges. For all $v_0 \in V$, we have that* $\texttt{find_trackable_vertex}(V, E, v_0)$ *outputs $(v, p)$ in polynomial time, where $p$ is a tracking path from $v_0$ to $v$.*

**Proof** Analyzing the time complexity using the typical strategy for a depth-first search, it is clear that the program terminates in $O(|V| + |E|)$ time, which is linear in the size of the input graph. As for correctness, it is invariant after the mutations on lines 2–4 and on lines 7–9 that $p$ is a path from $v_0$ to $v$, and that the set of marked vertices is the set of vertices contained in $p$. Additionally, upon reaching the return statement on line 11, the condition on line 6 must be false. That is, every outgoing neighbor of $v$ is marked. By the invariant, it follows that $p$ contains every outgoing neighbor of $v$. Thus, $p$ is a tracking path from $v_0$ to $v$. ■

Assured that one can find a trackable vertex, Theorem 9 demonstrates the previously outlined reduction. It follows that SSTIP is NP-complete, as in Corollary 10.

**Theorem 9** $\Gamma$-HTIP $\leq_{\mathrm{P}}$ $(\Gamma, \mathfrak{C})$-SSTIP *for all alphabets* $\Gamma$ *and structured congruence universes* $\mathfrak{C}$.

As for the notation in Algorithm 2, `join_labels` takes a path and outputs the string of its edge labels. `drop_until` takes a path $p$ and a vertex $v$ and outputs the path that is a suffix of $p$ and whose first vertex is $v$.

**Algorithm 2:** A mapping reduction from HTIP to SSTIP

1 **Function** $f_\sim((x_1, y_1), (x_2, y_2), \ldots (x_n, y_n))$**:**
2 $\quad([v]_\sim, p) \leftarrow$ `find_trackable_vertex`$(\Sigma^*/\sim, \{([u]_\sim, \sigma, [u\sigma]_\sim) \mid u \in \Sigma^*, \sigma \in \Sigma\}, [\lambda]_\sim)$;
3 $\quad$let $h : \Sigma \to \Sigma^*$ via $h(\sigma) \leftarrow \sigma \cdot$ `join_labels`(`drop_until`$(p, [v\sigma]_\sim)$);
4 $\quad$construct $t_h : \Sigma^* \to \Sigma^*$, the homomorphic transduction that $h$ recognizes;
5 $\quad$**return** $\{($`join_labels`$(p) \cdot t_h(x_i), y_i) \mid 1 \leq i \leq n\} \cup \{($`join_labels`$(p), \lambda)\}$;

**Proof** Let $\Gamma$ be an alphabet and let $\mathfrak{C}$ be a structured congruence universe with every $\mathfrak{C}_\Sigma$ having a canonical congruence called $\sim$. The desired reduction is given in Algorithm 2. The call to `find_trackable_vertex` runs in polynomial time by Lemma 8, and the rest of the program clearly runs in polynomial time. Also, the size of the instance mapped by $f_\sim$ is clearly polynomial in the size of the original instance. Recall from Lemma 8 that the input string `join_labels`$(p)$ traverses a tracking path from $[\lambda]_\sim$ to $[v]_\sim$ in $A_\sim$. As for correctness, we will show that $(\Sigma, (x_1, y_1), \ldots (x_n, y_n))$ is in $\Gamma$-HTIP if and only if $(\Sigma, f_\sim((x_1, y_1), \ldots (x_n, y_n)))$ is in $(\Gamma, \mathfrak{C})$-SSTIP.

First, assume that $(\Sigma, (x_1, y_1), \ldots (x_n, y_n))$ is in $\Gamma$-HTIP. Then, there is a total homomorphic transduction $t : \Sigma^* \to \Gamma^*$ such that $t(x_i) = y_i$ for all $i$. We construct the $\mathcal{C}$-subsequential transducer $T = \langle A_\sim, i, o, f\rangle$ according to Equation 2. Consider the $\mathcal{C}$-subsequential transduction $t' : \Sigma^* \to \Gamma^*$ that $T$ recognizes. Since the input string `join_labels`$(p)$ traverses a path to $[v]_\sim$, it cannot traverse any outgoing transition of $[v]_\sim$, so $o^*([\lambda]_\sim,$ `join_labels`$(p)) = \lambda$ and $t'($`join_labels`$(p)) = \lambda$. The same holds for every contained sub-traversal, so $o^*([v\sigma]_\sim,$ `join_labels`$($`drop_until`$(p, [v\sigma]_\sim))) = \lambda$ for all $\sigma \in \Sigma$. It then follows from Calculation 3 and $t(x_i) = y_i$ that $(\Sigma, f_\sim((x_1, y_1), \ldots (x_n, y_n)))$ is in $(\Gamma, \mathfrak{C})$-SSTIP with $t'$ as a witness.

$$\begin{aligned} i &= \lambda & o([v]_\sim, \sigma) &= t(\sigma) \\ f([u]_\sim) &= \lambda & o([u]_\sim, \sigma) &= \lambda \text{ for all } [u]_\sim \neq [v]_\sim \end{aligned} \tag{2}$$

Now, assume that $(\Sigma, f_\sim((x_1, y_1), \dots (x_n, y_n)))$ is in $(\Gamma, \mathfrak{C})$-SSTIP. Then, there is a total $\mathfrak{C}_\Sigma$-subsequential transduction $t' : \Sigma^* \to \Gamma^*$ such that $t'(\texttt{join_labels}(p) \cdot t_h(x_i)) = y_i$ and $t'(\texttt{join_labels}(p)) = \lambda$ for all $i$. Let $T = \langle A_\sim, i, o, f \rangle$ be a $\mathfrak{C}_\Sigma$-subsequential transducer that recognizes $t'$. It follows from $t'(\texttt{join_labels}(p)) = \lambda$ that $i = \lambda$, $o^*([\lambda]_\sim, \texttt{join_labels}(p)) = \lambda$, and $f([v]_\sim) = \lambda$. Since $o^*([\lambda]_\sim, \texttt{join_labels}(p)) = \lambda$, every contained sub-traversal must also produce the output $\lambda$, so $o^*([v\sigma]_\sim, \texttt{join_labels}(\texttt{drop_until}(p, [v\sigma]_\sim))) = \lambda$ for all $\sigma \in \Sigma$. We construct the homomorphic transduction $t : \Sigma^* \to \Gamma^*$ via $t(\sigma) = o([v]_\sim, \sigma)$. It follows from Calculation 3 and $t'(\texttt{join_labels}(p) \cdot t_h(x_i)) = y_i$ that $(\Sigma, (x_1, y_1), \dots (x_n, y_n))$ is in $\Gamma$-HTIP with $t$ as a witness.

$$
\begin{aligned}
t'(\texttt{join_labels}(p) \cdot t_h(x_i)) &= i \cdot o^*([\lambda]_\sim, \texttt{join_labels}(p) \cdot t_h(x_i)) \cdot f(\cdots) \\
&= o^*([\lambda]_\sim, \texttt{join_labels}(p)) \cdot \\
&\qquad o^*(\delta^*_\sim([\lambda]_\sim, \texttt{join_labels}(p)), t_h(x_i)) \\
&= o^*([v]_\sim, t_h(x_i)) \\
&= \textstyle\prod_{\text{prefixes } u\sigma \text{ from } x_i} o^*(\delta^*_\sim([v]_\sim, t_h(u)), t_h(\sigma)) \\
&= \textstyle\prod_{\sigma \text{ from } x_i} o^*([v]_\sim, h(\sigma)) \\
&= \textstyle\prod_{\sigma \text{ from } x_i} [o([v]_\sim, \sigma) \cdot \\
&\qquad o^*([v\sigma]_\sim, \texttt{join_labels}(\texttt{drop_until}(p, [v\sigma]_\sim)))] \\
&= \textstyle\prod_{\sigma \text{ from } x_i} t(\sigma) \\
&= t(x_i)
\end{aligned}
\tag{3}
$$

■

**Corollary 10** $(\Gamma, \mathfrak{C})$-SSTIP *is* NP*-complete for all alphabets* $\Gamma$ *where* $|\Gamma| \geq 2$ *and structured congruence universes* $\mathfrak{C}$*.*

**Proof** It follows from Lemma 2 and Theorem 9 that $(\Gamma, \mathfrak{C})$-SSTIP is NP-hard. Since $(\Gamma, \mathfrak{C})$-SSTIP is also in NP from Lemma 6, we may conclude that $(\Gamma, \mathfrak{C})$-SSTIP is NP-complete. ■

## 5. Reducing SSTIP to HTIP

Theorem 11 demonstrates a straightforward reduction in the other direction, from SSTIP to HTIP. From here, it is easy to derive a solver for SSTIP by encoding HTIP into the language of any sufficiently expressive NP string theory solver. For instance, one could employ the string theory SAT encoding given by Lotz et al. (2024).

The reduction works by mapping each input string $x$ to the string of transitions traversed by a $\mathcal{C}$-subsequential transducer processing $x$. This way, a witness homomorphic transducer for the mapped HTIP instance is exactly the assignment of transitions to output strings that characterizes a witness $\mathcal{C}$-subsequential transducer for the original SSTIP instance.

Let $\sim$ be the canonical congruence of $\mathcal{C}$. In the mapped string of transitions, the initial transition will be represented by the left delimiter $\langle \rtimes \rangle$. Transitions between states will be

represented by a pair $\langle [u]_\sim, \sigma \rangle$ of the source state $[u]_\sim$ and the character $\sigma$. Final transitions will be represented by a pair $\langle [u]_\sim, \ltimes \rangle$ of the final state $[u]_\sim$ and the right delimiter $\ltimes$. Overall, the alphabet of transitions is $\{\rtimes\} \cup ((\Sigma^*/\sim) \times (\Sigma \cup \{\ltimes\}))$ and will be called $\hat{\Sigma}$.

**Theorem 11** $(\Gamma, \mathfrak{C})$-SSTIP $\leq_{\mathrm{P}}$ $\Gamma$-HTIP *for all alphabets* $\Gamma$ *and structured congruence universes* $\mathfrak{C}$.

**Algorithm 3:** A mapping reduction from SSTIP to HTIP

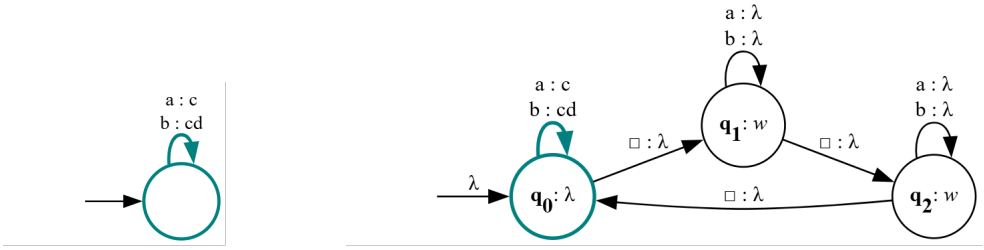
**Function** $f_\sim((x_1, y_1), (x_2, y_2), \ldots (x_n, y_n))$**:**
- let $i_T \in \hat{\Sigma}^*$ via $i_T \leftarrow \langle \rtimes \rangle$;
- let $o_T : (\Sigma^*/\sim) \times \Sigma \to \hat{\Sigma}^*$ via $o_T([u]_\sim, \sigma) \leftarrow \langle [u]_\sim, \sigma \rangle$;
- let $f_T : (\Sigma^*/\sim) \to \hat{\Sigma}^*$ via $f_T([u]_\sim) \leftarrow \langle [u]_\sim, \ltimes \rangle$;
- $T \leftarrow \langle A_\sim, i_T, o_T, f_T \rangle$;
- construct $t_T : \Sigma^* \to \hat{\Sigma}^*$, the $\mathfrak{C}_\Sigma$-subsequential transduction that $T$ recognizes;
- **return** $\{(t_T(x_i), y_i) \mid 1 \leq i \leq n\}$;

**Proof** The desired reduction is given in Algorithm 3. Since the proof is not much more enlightening than the previous reduction outline and does not apply the simulation strategy that is the focus of this paper, it will be deferred to Appendix A. ■

## 6. Identifying Bounded Classes of Subsequential Transductions

This section applies the simulation strategy to reduce HTIP to BSTIP, proving that BSTIP is NP-complete, as in Corollary 13. The reduction is demonstrated in Theorem 12 and works by isolating the initial state in the $m$-state transducer around which to simulate a single-state machine, that is, a homomorphic transducer. This is similar to the simulation from the reduction in Section 4, except that in this case, the transition structure of the transducer is not known a priori. Instead, this reduction inserts data that enforces a desired transition structure. In our desired transition structure, every state other than the initial state is rendered *useless* by forcing the concatenation of an *out-of-bounds* string to the output if traversed. This way, it is guaranteed that the input strings from the original data only ever traverse the initial state.

More precisely, the reduction is as follows. A character $\square$ is added to $\Sigma$ to traverse the useless states without using characters in the original data. The out-of-bounds string $w$ is chosen to be longer than any output string in the original data, guaranteeing that it never appears as a substring. The states are arranged so that the $\square$ transitions cycle all $m$ states by inserting the data $(\square^j, w)$ for all $1 \leq j < m$, and $(\square^m, \lambda)$. This also ensures that the useless states have $w$ as their final output string, and that the initial output string and the final output string of the initial state are both $\lambda$. The outgoing transitions of the useless states are accounted for by inserting the data $(\square^j \cdot \sigma, w)$ for all $1 \leq j < m$, and $\sigma \in \Sigma$. This way, it is impossible to traverse to a useless state and back to the initial state without encountering the output string $w$. Figure 3 visualizes this reduction outline with $m \leftarrow 3$.




$$\texttt{data} = \{(a, c), (ab, ccd), (ba, cdc)\}$$

$$f_{3,\gamma}(\texttt{data}) = \{(a, c), (ab, ccd), (ba, cdc), (\square, w), (\square a, w), (\square b, w), (\square\square, w), (\square\square a, w), (\square\square b, w), (\square\square\square, \lambda)\}$$

(a) An HTIP instance and a witness (b) The mapped BSTIP instance and witness

Figure 3: The HTIP witness is simulated around the $q_0$ state in the BSTIP witness.

**Algorithm 4:** A mapping reduction from HTIP to BSTIP

**Function** $f_{m,\gamma}((x_1, y_1), (x_2, y_2), \ldots (x_n, y_n))$**:**
  $k \leftarrow 1 + \max_{i=1}^{n} |y_i|$;
  $w \leftarrow \gamma^k$;
  **return**
    $\{(x_i, y_i) \mid 1 \leq i \leq n\} \cup$
    $\{(\square^j, w) \mid 1 \leq j < m\} \cup$
    $\{(\square^j \cdot \sigma, w) \mid 1 \leq j < m, \sigma \in \Sigma\} \cup$
    $\{(\square^m, \lambda)\}$;

**Theorem 12** $\Gamma$-HTIP $\leq_{\mathrm{P}}$ $(\Gamma, m)$-BSTIP *for all alphabets* $\Gamma$ *and* $m \in \mathbb{N}^+$.

**Proof** Let $\Gamma$ be an alphabet, let $m \in \mathbb{N}^+$, and take $\gamma \in \Gamma$. The desired reduction is given in Algorithm 4. The program clearly runs in polynomial time. Also, the size of the instance mapped by $f_{m,\gamma}$ is clearly polynomial in the size of the original instance. As for correctness, we will show that $(\Sigma, (x_1, y_1), \ldots (x_n, y_n))$ is in $\Gamma$-HTIP if and only if $(\Sigma \cup \{\square\}, f_{m,\gamma}((x_1, y_1), \ldots (x_n, y_n)))$ is in $(\Gamma, m)$-BSTIP.

First, assume that $(\Sigma, (x_1, y_1), \ldots (x_n, y_n))$ is in $\Gamma$-HTIP. Then, there is a total homomorphic transduction $t : \Sigma^* \to \Gamma^*$ such that $t(x_i) = y_i$ for all $i$. We construct the $m$-state transducer $T = \langle A, i, o, f \rangle$ according to Equation 4. Consider the $m$-state transduction $t' : (\Sigma \cup \{\square\})^* \to \Gamma^*$ that $T$ recognizes. It follows from Calculation 7 (Appendix B), Calculation 5, and $t(x_i) = y_i$ that $(\Sigma \cup \{\square\}, f_{m,\gamma}((x_1, y_1), \ldots (x_n, y_n)))$ is in $(\Gamma, m)$-BSTIP with

$t'$ as a witness.

$$\begin{aligned} A &= \langle \mathbb{Z}/m\mathbb{Z}, q_0, \delta, \mathbb{Z}/m\mathbb{Z} \rangle & \delta(j, \square) &= j+1 \\ q_0 &= 0 & \delta(j,\sigma) &= j \text{ for all } \sigma \in \Sigma \\ \\ i &= \lambda & o(j, \square) &= \lambda \\ f(q_0) &= \lambda & o(q_0, \sigma) &= t(\sigma) \text{ for all } \sigma \in \Sigma \\ f(j) &= w \text{ for all } j \neq q_0 & o(j,\sigma) &= \lambda \text{ for all } j \neq q_0 \text{ and } \sigma \in \Sigma \end{aligned} \tag{4}$$

Now, assume that $(\Sigma \cup \{\square\}, f_{m,\gamma}, ((x_1, y_1), \dots (x_n, y_n)))$ is in $(\Gamma, m)$-BSTIP. Then, there is a total $m$-state transduction $t' : (\Sigma \cup \{\square\})^* \to \Gamma^*$ such that $t'(x_i) = y_i$, $t'(\square^j) = w$, $t'(\square^j \cdot \sigma) = w$, and $t'(\square^m) = \lambda$ for all $i$, $1 \leq j < m$, and $\sigma \in \Sigma$. Let $T = \langle A, i, o, f \rangle$ be an $m$-state transducer that recognizes $t'$. It follows from $t'(\square^m) = \lambda$ that $i = \lambda$, $o(\delta^*(q_0, \square^j), \square) = \lambda$, and $f(\delta^*(q_0, \square^m)) = \lambda$ for all $0 \leq j < m$, since $t'(\square^m)$ would be nonempty if any of those output strings were nonempty. It then follows from $t'(\square^j) = w$ that $f(\delta^*(q_0, \square^j)) = w$ for all $1 \leq j < m$, since the initial output string and the transition output strings traversed by $t'(\square^j)$ have been proven empty, so the output $w$ can only be the final output string.

We will now show that the $\square$ transitions traverse all $m$ states before completing a cycle, that is, $\delta^*(q_0, \square^j) = \delta^*(q_0, \square^k) \implies j = k$ for all $0 \leq j, k < m$. Assume for the sake of contradiction that there exist $0 \leq j, k < m$ such that $\delta^*(q_0, \square^j) = \delta^*(q_0, \square^k)$ but $j < k$. Then, $1 \leq k - j < m$, so $1 \leq m - (k - j) < m$. Recall that $\lambda = f(\delta^*(q_0, \square^m))$, and now $f(\delta^*(q_0, \square^m)) = f(\delta^*(\delta^*(q_0, \square^k), \square^{m-k})) = f(\delta^*(\delta^*(q_0, \square^j), \square^{m-k})) = f(\delta^*(q_0, \square^{m-(k-j)}))$, and observe that $f(\delta^*(q_0, \square^{m-(k-j)})) = w$. Thus, $\lambda = w$, which is the desired contradiction. Moreover, the cycle is completed by traversing back to the initial state, that is, $\delta^*(q_0, \square^m) = q_0$. This is because there are only $m$ states, so every other state is one from the cycle $\delta^*(q_0, \square^j)$ where $1 \leq j < m$. Thus, traversing back to one would entail the same contradiction $\lambda = w$.

We will now show that after traversing to a non-initial state, it is impossible to avoid producing the output $w$ without traversing $\square$ transitions, that is, $o^*(\delta^*(q_0, \square^j), u) \cdot f(\delta^*(q_0, \square^j \cdot u))$ has $w$ as a substring for all $1 \leq j < m$ and $u \in \Sigma^*$. It follows from $t'(\square^j \cdot \sigma) = w$ that $\delta^*(q_0, \square^j \cdot \sigma) = q_0 \implies o(\delta^*(q_0, \square^j), \sigma) = w$ for all $1 \leq j < m$ and $\sigma \in \Sigma$, since the initial output string and the $\square$ transition output strings traversed by $t'(\square^j \cdot \sigma)$ have been proven empty, so if the final output string is also empty, then the output $w$ can only be the $\sigma$ transition output string. By induction on the length of $u$ with base case $u \leftarrow \lambda$, it is easy to extend this result to $\delta^*(q_0, \square^j \cdot u) = q_0 \implies$ "$o^*(\delta^*(q_0, \square^j), u)$ has $w$ as a substring" for all $1 \leq j < m$ and $u \in \Sigma^*$. Now for the desired result, let $1 \leq j < m$ and $u \in \Sigma^*$. In the case that $\delta^*(q_0, \square^j \cdot u) = q_0$, we get $o^*(\delta^*(q_0, \square^j), u)$ has $w$ as a substring. In the case that $\delta^*(q_0, \square^j \cdot u) \neq q_0$, we get $f(\delta^*(q_0, \square^j \cdot u)) = w$. Either way, $o^*(\delta^*(q_0, \square^j), u) \cdot f(\delta^*(q_0, \square^j \cdot u))$ has $w$ as a substring as desired.

Finally, we can show that $\delta(q_0, \sigma) = q_0$ for all $\sigma$ from any $x_i$. Assume for the sake of contradiction that there exist $\sigma_1, \sigma_2, \dots \sigma_c$ from $x_i$ such that $\delta(q_0, \sigma) = \delta^*(q_0, \square^j)$ for some $1 \leq j < m$ if and only if $\sigma = \sigma_b$ for some $1 \leq b \leq c$. Let $v$ be the longest prefix of $x_i$ that does not contain any of $\sigma_1, \sigma_2, \dots \sigma_c$, that is, $v \in (\Sigma \setminus \{\sigma_1, \sigma_2, \dots \sigma_c\})^*$ and $u \in \Sigma^*$ such that $v\sigma_b u = x$ for some $1 \leq b \leq c$. Then by Calculation 8 (Appendix B), $y_i = t'(x_i)$

has $o^*(\delta^*(q_0, \square^j), u) \cdot f(\delta^*(q_0, \square^j \cdot u))$ as a substring, which has $w$ as a substring. This is the desired contradiction, since $w$ is longer than any $y_i$. We construct the homomorphic transduction $t : \Sigma^* \to \Gamma^*$ via $t(\sigma) = o(q_0, \sigma)$. It follows from Calculation 5 and $t'(x_i) = y_i$ that $(\Sigma, (x_1, y_1), \ldots (x_n, y_n))$ is in $\Gamma$-HTIP with $t$ as a witness.

$$\begin{aligned} t'(x_i) &= i \cdot o^*(q_0, x_i) \cdot f(\delta^*(q_0, x_i)) \\ &= \left[\prod_{\text{prefixes } u\sigma \text{ from } x_i} o(\delta^*(q_0, u), \sigma)\right] \cdot f(\delta^*(q_0, x_i)) \\ &= \left[\prod_{\sigma \text{ from } x_i} o(q_0, \sigma)\right] \cdot f(q_0) \\ &= \prod_{\sigma \text{ from } x_i} t(\sigma) \\ &= t(x_i) \end{aligned} \tag{5}$$

■

**Corollary 13** $(\Gamma, m)$-BSTIP *is* NP*-complete for all alphabets* $\Gamma$ *where* $|\Gamma| \geq 2$ *and* $m \in \mathbb{N}^+$.

**Proof** It follows from Lemma 2 and Theorem 12 that $(\Gamma, m)$-BSTIP is NP-hard. Since $(\Gamma, m)$-BSTIP is also in NP from Lemma 4, we may conclude that $(\Gamma, m)$-BSTIP is NP-complete. ■

## 7. Conclusion

This paper proposes the *simulation strategy* for proving the hardness of transducer identification problems. We apply the strategy to prove that the identification of structured classes of subsequential transductions is NP-complete, which implies that SOSFIA's (Jardine et al., 2014) feasibility limitation cannot be addressed by designing a feasible learner unless P = NP. We demonstrate that the strategy gives a simpler proof that identification of bounded classes of subsequential transductions is NP-complete, which is the subsequential transducer version of a known result for Mealy-model transducers by Gold (1978). We also provide a reduction from the identification of structured classes of subsequential transductions to the identification of homomorphic transductions, a problem that can be encoded into the language of current NP string theory solvers (Lotz et al., 2024).

In future work, the simulation strategy could be applied to more classes of transductions than subsequential ones. Currently, the hardness of identifying rational transducers is less understood than that of identifying subsequential transducers. Yet, both NP-complete identification problems in this paper, SSTIP and BSTIP, have natural rational counterparts. We hypothesize that both rational transducer identification problems can be proven NP-complete using the strategy proposed in this paper, especially considering the bimachine representation of $\mathcal{C}$-rational transductions due to Filiot et al. (2019).

## Acknowledgments

I would like to thank the reviewers for their constructive comments, which helped to clarify the representation of structured congruence universes in the current version of the paper. I would also like to thank those who gave helpful feedback on earlier versions of this work.

## References


Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987. doi: 10.1016/0890-5401(87)90052-6. URL https://doi.org/10.1016/0890-5401(87)90052-6.

Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars*, chapter 7, pages 143–172. Cambridge University Press, 2010. ISBN 9780521763165. URL https://doi.org/10.1017/CBO9781139194655.

Andrzej Ehrenfeucht and Grzegorz Rozenberg. Finding a homomorphism between two words is np-complete. *Information Processing Letters*, 9(2):86–88, 1979. doi: 10.1016/0020-0190(79)90135-2. URL https://doi.org/10.1016/0020-0190(79)90135-2.

Emmanuel Filiot, Olivier Gauwin, and Nathan Lhote. Logical and algebraic characterizations of rational transductions. *Logical Methods in Computer Science*, 15(4), 2019. doi: 10.23638/LMCS-15(4:16)2019. URL https://doi.org/10.23638/LMCS-15(4:16)2019.

Daniel Gildea and Daniel Jurafsky. Learning bias and phonological-rule induction. *Computational Linguistics*, 22(4):497–530, 1996.

E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967. doi: 10.1016/S0019-9958(67)91165-5. URL https://doi.org/10.1016/S0019-9958(67)91165-5.

E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978. doi: 10.1016/S0019-9958(78)90562-4. URL https://doi.org/10.1016/S0019-9958(78)90562-4.

Adam Jardine, Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. Very efficient learning of structured classes of subsequential functions from positive data. In Alexander Clark, Makoto Kanazawa, and Ryo Yoshinaka, editors, *Proceedings of the 12th International Conference on Grammatical Inference, ICGI 2014, Kyoto, Japan, September 17-19, 2014*, JMLR Workshop and Conference Proceedings, pages 94–108. JMLR.org, 2014. URL http://proceedings.mlr.press/v34/jardine14a.html.

Kevin Lotz, Amit Goel, Bruno Dutertre, Benjamin Kiesl-Reiter, Soonho Kong, and Dirk Nowotka. Solving string constraints with concatenation using SAT. In Nina Narodytska and Philipp Rümmer, editors, *Formal Methods in Computer-Aided Design, FMCAD 2024, Prague, Czech Republic, October 15-18, 2024*, pages 29–38. IEEE, 2024. doi: 10.34727/2024/ISBN.978-3-85448-065-5_9. URL https://doi.org/10.34727/2024/isbn.978-3-85448-065-5_9.

José Oncina, Pedro García, and Enrique Vidal. Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):448–458, 1993. doi: 10.1109/34.211465. URL https://doi.org/10.1109/34.211465.

Michael Sipser. *Introduction to the theory of computation*, chapter 7, pages 275–330. Cengage Learning, 3rd edition, 2013.

## Appendix A. Proof of Theorem 11

**Proof** Let $\Gamma$ be an alphabet and let $\mathfrak{C}$ be a structured congruence universe with every $\mathfrak{C}_\Sigma$ having a canonical congruence called $\sim$. The desired reduction is given in Algorithm 3. The program clearly runs in polynomial time. Also, the size of the instance mapped by $f_\sim$ is clearly polynomial in the size of the original instance. As for correctness, we will show that $(\Sigma, (x_1, y_1), \ldots (x_n, y_n))$ is in $(\Gamma, \mathfrak{C})$-SSTIP if and only if $(\hat{\Sigma}, f_\sim((x_1, y_1), \ldots (x_n, y_n)))$ is in $\Gamma$-HTIP.

First, assume that $(\Sigma, (x_1, y_1), \ldots (x_n, y_n))$ is in $(\Gamma, \mathfrak{C})$-SSTIP. Then, there is a total $\mathfrak{C}_\Sigma$-subsequential transduction $t' : \Sigma^* \to \Gamma^*$ such that $t'(x_i) = y_i$ for all $i$. Let $T = \langle A_\sim, i, o, f\rangle$ be a $\mathfrak{C}_\Sigma$-subsequential transducer that recognizes $t'$. We construct the homomorphic transduction $t : \hat{\Sigma}^* \to \Gamma^*$ via $t(\langle \rtimes \rangle) = i$, $t(\langle [u]_\sim, \sigma\rangle) = o([u]_\sim, \sigma)$, and $t(\langle [u]_\sim, \ltimes\rangle) = f([u]_\sim)$. It follows from Calculation 6 and $t'(x_i) = y_i$ that $(\hat{\Sigma}, f_\sim((x_1, y_1), \ldots (x_n, y_n)))$ is in $\Gamma$-HTIP with $t$ as a witness.

Now, assume that $(\hat{\Sigma}, f_\sim((x_1, y_1), \ldots (x_n, y_n)))$ is in $\Gamma$-HTIP. Then, there is a total homomorphic transduction $t : \hat{\Sigma}^* \to \Gamma^*$ such that $t(t_T(x_i)) = y_i$ for all $i$. We construct the $\mathfrak{C}_\Sigma$-subsequential transducer $T = \langle A_\sim, i, o, f\rangle$ where $i = t(\langle \rtimes \rangle)$, $o([u]_\sim, \sigma) = t(\langle [u]_\sim, \sigma\rangle)$, and $f([u]_\sim) = t(\langle [u]_\sim, \ltimes\rangle)$. Consider the $\mathfrak{C}_\Sigma$-subsequential transduction $t' : \Sigma^* \to \Gamma^*$ that $T$ recognizes. It follows from Calculation 6 and $t(t_T(x_i)) = y_i$ that $(\Sigma, (x_1, y_1), \ldots (x_n, y_n))$ is in $(\Gamma, \mathfrak{C})$-SSTIP with $t'$ as a witness.

$$
\begin{aligned}
t(t_T(x_i)) &= t(i_T \cdot o_T^*([\lambda]_\sim, x_i) \cdot f_T(\delta_\sim^*([\lambda]_\sim, x_i))) \\
&= t(i_T \cdot \left[\prod_{\text{prefixes } u\sigma \text{ from } x_i} o_T(\delta_\sim^*([\lambda]_\sim, u), \sigma)\right] \cdot f_T(\delta_\sim^*([\lambda]_\sim, x_i))) \\
&= t(\langle \rtimes \rangle \cdot \left[\prod_{\text{prefixes } u\sigma \text{ from } x_i} \langle \delta_\sim^*([\lambda]_\sim, u), \sigma\rangle\right] \cdot \langle \delta_\sim^*([\lambda]_\sim, x_i), \ltimes\rangle) \\
&= t(\langle \rtimes \rangle) \cdot \left[\prod_{\text{prefixes } u\sigma \text{ from } x_i} t(\langle \delta_\sim^*([\lambda]_\sim, u), \sigma\rangle)\right] \cdot t(\langle \delta_\sim^*([\lambda]_\sim, x_i), \ltimes\rangle) \qquad (6) \\
&= i \cdot \left[\prod_{\text{prefixes } u\sigma \text{ from } x_i} o(\delta_\sim^*([\lambda]_\sim, u), \sigma)\right] \cdot f(\delta_\sim^*([\lambda]_\sim, x_i)) \\
&= i \cdot o^*([\lambda]_\sim, x_i) \cdot f(\delta_\sim^*([\lambda]_\sim, x_i)) \\
&= t'(x_i)
\end{aligned}
$$

■

## Appendix B. Calculations for Theorem 12

$$\begin{aligned} t'(\square^j) &= i \cdot o^*(0, \square^j) \cdot f(\delta^*(0, \square^j)) \\ &= f(j) = w \qquad (\text{by } 1 \le j < m) \end{aligned}$$

$$\begin{aligned} t'(\square^j \cdot \sigma) &= i \cdot o^*(0, \square^j \cdot \sigma) \cdot f(\delta^*(0, \square^j \cdot \sigma)) \\ &= o^*(0, \square^j) \cdot o(\delta^*(0, \square^j), \sigma) \cdot f(\delta(\delta^*(0, \square^j), \sigma)) \\ &= o(j, \sigma) \cdot f(\delta(j, \sigma)) \\ &= f(j) = w \qquad (\text{by } 1 \le j < m) \end{aligned} \tag{7}$$

$$\begin{aligned} t'(\square^m) &= i \cdot o^*(0, \square^m) \cdot f(\delta^*(0, \square^m)) \\ &= f(m) = f(0) = \lambda \qquad (\text{by } m = 0 \in \mathbb{Z}/m\mathbb{Z}) \end{aligned}$$

$$\begin{aligned} t'(x_i) = t'(v\sigma_b u) &= i \cdot o^*(q_o, v\sigma_b u) \cdot f(\delta^*(q_0, v\sigma_b u)) \\ &= o^*(q_0, v\sigma_b) \cdot o^*(\delta(\delta^*(q_0, v), \sigma_b), u) \cdot f(\delta^*(\delta(\delta^*(q_0, v), \sigma_b), u)) \\ &= o^*(q_0, v\sigma_b) \cdot o^*(\delta(q_0, \sigma_b), u) \cdot f(\delta^*(\delta(q_0, \sigma_b), u)) \\ &= o^*(q_0, v\sigma_b) \cdot o^*(\delta^*(q_0, \square^j), u) \cdot f(\delta^*(\delta^*(q_0, \square^j), u)) \\ &= o^*(q_0, v\sigma_b) \cdot o^*(\delta^*(q_0, \square^j), u) \cdot f(\delta^*(q_0, \square^j \cdot u)) \end{aligned} \tag{8}$$