

What You Must Remember When Transforming Datawords

M. Praveen

Chennai Mathematical Institute, India
UMI ReLaX

Abstract

Streaming Data String Transducers (SDSTs) were introduced to model a class of imperative and a class of functional programs, manipulating lists of data items. These can be used to write commonly used routines such as insert, delete and reverse. SDSTs can handle data values from a potentially infinite data domain. The model of Streaming String Transducers (SSTs) is the fragment of SDSTs where the infinite data domain is dropped and only finite alphabets are considered. SSTs have been much studied from a language theoretical point of view. We introduce data back into SSTs, just like data was introduced to finite state automata to get register automata. The result is Streaming String Register Transducers (SSRTs), which is a subclass of SDSTs. SDSTs can compare data values using a linear order on the data domain, which can't be done by SSRTs.

We give a machine independent characterization of SSRTs with origin semantics, along the lines of Myhill-Nerode theorem. Machine independent characterizations for similar models have formed the basis of learning algorithms and enabled us to understand fragments of the models. Origin semantics of transducers track which positions of the output originate from which positions of the input. Although a restriction, using origin semantics is well justified and known to simplify many problems related to transducers. We use origin semantics as a technical building block, in addition to characterizations of deterministic register automata. However, we need to build more on top of these to overcome some challenges unique to SSRTs.

2012 ACM Subject Classification Theory of computation → Transducers; Theory of computation → Automata over infinite objects

Keywords and phrases Streaming String Transducers, Data words, Machine independent characterization

Funding *M. Praveen*: Partially supported by a grant from the Infosys foundation.

Acknowledgements The author thanks C. Aishwarya and anonymous reviewers for suggestions to improve presentation and pointers to related works.

1 Introduction

Transductions are in general relations among words over an input alphabet and words over an output alphabet. Transducers are theoretical models that implement transductions. Transducers are used for a variety of applications, such as analysis of web sanitization frameworks, host based intrusion detection, natural language processing, modeling some classes of programming languages and constructing programming language tools like evaluators, type checkers and translators. Streaming Data String Transducers (SDSTs) were introduced in [1] to model a class of imperative and a class of functional programs, manipulating lists of data items. Transducers have been used in [11] to infer semantic interfaces of data structures such as stacks. Such applications use Angluin style learning, which involves constructing transducers by looking at example operations of the object under study. Since the transducer is still under construction, we need to make inferences about the transduction without having access to a transducer that implements the transduction. Theoretical bases for doing this are machine independent characterizations, which identify what kind of transductions can be implemented by what kind of transducers and give a template for constructing transducers.

Indeed the seminal Myhill-Nerode theorem gives a machine independent characterization for regular languages over finite alphabets, which form the basis of Angluin style learning of regular languages [2]. A similar characterization for a fragment of SDSTs is given in [4] and used as a basis to design a learning algorithm.

Programs deal with data from an infinite domain and transducers modeling the programs should also treat data as such. Indeed, it is mentioned in [11] that inferring a data structure like “stack of stacks” presuming a data domain of 4 elements resulted in a transducer of more than 10^9 states and would require billions of queries. On the other hand, using a transducer model that can deal with data from an infinite domain resulted in a transducer with roughly 800 states and needed 4000 queries. The transducers used in [11] produce the output in a linear fashion without remembering what was output before. For example, they can not output the reverse of the input strings. SDSTs are more powerful and can implement a bigger class of transductions, including reversal. However, formal language theoretic studies of this model have been restricted to Streaming String Transducers (SSTs), which don’t deal with data values from an infinite domain. In this paper, we add the ability to deal with data values back into SSTs, resulting in Streaming String Register Transducers (SSRTs) and give a machine independent characterization using origin semantics. SSRTs are not as powerful as the original SDSTs, since SSRTs can not compare data values using linear orders, which SDSTs can do. This is mainly to keep technical tediousness of proofs manageable.

Origin semantics of transducers were introduced in [4], which considers how positions of the output originate from the positions of the input. Using origin semantics alters the answers to some of the questions about transducers. E.g., two transducers that may be equivalent according to the classical semantics may not be equivalent according to origin semantics. Origin semantics is a reasonable restriction and is used extensively in this paper.

Apart from providing a basis for Angluin style learning algorithms, machine independent characterizations are also useful for studying fragments of transducer models. For example, in [4], machine independent characterization of SSTs has been used to study fragments such as non-deterministic automata with output and transductions definable in First Order logic.

Related Works

It is shown in [1] that checking equivalence of SDSTs is in PSPACE. A machine independent characterization for SSTs is given in [4]. Origin semantics of transductions and factored outputs are key concepts introduced in that paper that we use here. However, SSTs do not handle data values and we need more ideas here. SSTs use variables to store intermediate values while computing transductions; this idea appears in an earlier work [8], which introduced *simple programs on strings*, which implement the same class of transductions as those implemented by SSTs. Machine independent characterizations for automata with finite memory are given in [10, 3]. Formalizations of what data values need to be kept in memory by automata are key concepts introduced in those papers that we use here. However, the models considered there are automata that only accept or reject their input. We need more work here since we deal with transducers that produce output. Angluin style learning algorithm for deterministic automata with memory is given in [12]. Again this is only for automata that accept or reject their inputs. A machine independent characterization of automata with finite memory is given in [5], which is further extended to data domains with arbitrary binary relations in [6]. These papers also formalize which data values need to be kept in memory, but only consider automata as opposed to transducers. The learning algorithm of [12] is extended to Mealy machines with data in [11]. However, Mealy machines are not as powerful as SSRTs that we consider here. Using a more abstract approach of

nominal automata, [14] presents a learning algorithm for automata over infinite alphabets. Logical characterizations of transducers that can handle data are considered in [9]. However, the transducers in that paper can not use data values to make decisions, although they are part of the output. Register automata with linear arithmetic introduced in [7] shares some of the features of the transducer model used here. Here, data words stored in variables can be concatenated, while in register automata with linear arithmetic, numbers stored in variables can be operated upon by linear operators.

Proofs of some of the results in this paper are tedious and are moved to the appendix to maintain flow of ideas in the main paper. Proofs of results stated in the main part of the paper are in Sections B, C and D. Section A states and proves some basic properties of transductions and transducers that are only invoked in Sections B, C and D. Section E contains proofs that are especially long. It consists of lengthy case analysis to rigorously verify facts that are intuitively clear.

2 Preliminaries

Let \mathbb{I} be the set of integers, \mathbb{N} be the set of non-negative integers and D be an infinite set of data values. We will refer to D as the *data domain*. For $i, j \in \mathbb{I}$, we denote by $[i, j]$ the set $\{k \mid i \leq k \leq j\}$. For any set S , S^* denotes the set of all finite sequences of elements from S . The empty sequence is denoted by ϵ . Given $u, v \in S^*$, v is a *prefix* (resp. *suffix*) of u if there exists $w \in S^*$ such that $u = vw$ (resp. $u = wv$). The sequence v is an *infix* of u if there are sequences w_1, w_2 such that $u = w_1vw_2$.

Let Σ, Γ be finite alphabets. We will use Σ for input alphabet and Γ for output alphabet. A *data word* over Σ is a word in $(\Sigma \times D)^*$. A *data word with origin information* over Γ is a word in $(\Gamma \times D \times \mathbb{N})^*$. Suppose $\Sigma = \{\text{title}, \text{firstName}, \text{lastName}\}$ and $\Gamma = \{\text{givenName}, \text{surName}\}$. An example data word over Σ is $(\text{title}, \text{Mr.})(\text{firstName}, \text{Harry})(\text{lastName}, \text{Tom})$. If we were to give this as input to a device that reverses the order of names, the output would be the data word with origin information $(\text{surName}, \text{Tom}, 3)(\text{givenName}, \text{Harry}, 2)$, over Γ . In the triple $(\text{givenName}, \text{Harry}, 2)$, the third component 2 indicates that the pair $(\text{givenName}, \text{Harry})$ originates from the second position of the input data word. We call the third component *origin* and it indicates the position in the input that is responsible for producing the output triple. The data value at some position of the output may come from any position (not necessarily the origin) of the input data word. We write *transduction* for any function from data words over Σ to data words with origin information over Γ .

For a data word w , $|w|$ is its length. For a position $i \in [1, |w|]$, we denote by $\text{data}(w, i)$ (resp. $\text{letter}(w, i)$) the data value (resp. the letter from the finite alphabet) at the i^{th} position of w . We denote by $\text{data}(w, *)$ the set of all data values that appear in w . For positions $i \leq j$, we denote by $w[i, j]$ the infix of w starting at position i and ending at position j . Note that $w[1, |w|] = w$. Two data words w_1, w_2 are *isomorphic* (denoted by $w_1 \simeq w_2$) if $|w_1| = |w_2|$, $\text{letter}(w_1, i) = \text{letter}(w_2, i)$ and $\text{data}(w_1, i) = \text{data}(w_1, j)$ iff $\text{data}(w_2, i) = \text{data}(w_2, j)$ for all positions $i, j \in [1, |w_1|]$. For data values d, d' , we denote by $w[d/d']$ the data word obtained from w by replacing all occurrences of d by d' . We say that d' is a *safe replacement* for d in w if $w[d/d'] \simeq w$. Intuitively, replacing d by d' doesn't introduce new equalities/inequalities among the positions of w . For example, d_1 is a safe replacement for d_2 in $(a, d_3)(b, d_2)$, but not in $(a, d_1)(b, d_2)$.

A permutation on data values is any bijection $\pi : D \rightarrow D$. For a data word u , $\pi(u)$ is obtained from u by replacing all its data values by their respective images under π . A transduction f is *invariant under permutations* if for every data word u and every permutation

$\pi, f(\pi(u)) = \pi(f(u))$ (permutation can be applied before or after the transduction).

Suppose a transduction f has the property that for any triple (γ, d, o) in any output $f(w)$, there is a position $i \leq o$ in w such that $\text{data}(w, i) = d$. Intuitively, if the data value d is output from the origin o , then d should have already occurred in the input on or before o . Such transductions are said to be *without data peeking*. We say that a transduction has *linear blow up* if there is a constant K such that for any position o of any input, there are at most K positions in the output whose origin is o .

Streaming String Register Transducers

We extend SSTs to handle data values, just like finite state automata were extended to finite memory automata [13]. The result is a subclass of deterministic Streaming Data String Transducers (SDSTs), introduced in [1]. SDSTs can compare data values using a linear order on the data domain D , but we allow comparison for only equality in our model.

► **Definition 1.** A Streaming String Register Transducer (SSRT) is an eight tuple $S = (\Sigma, \Gamma, Q, q_0, R, X, O, \Delta)$, where

- the finite alphabets Σ, Γ are used for input, output respectively,
- Q is a finite set of states, q_0 is the initial state,
- R is a finite set of registers and X is a finite set of data word variables,
- $O : Q \rightarrow ((\Gamma \times \hat{R}) \cup X)^*$ is a partial output function, where $\hat{R} = R \cup \{\text{curr}\}$, with *curr* being a special symbol used to denote the current data value being read and
- $\Delta \subseteq (Q \times \Sigma \times \Phi \times Q \times 2^R \times U)$ is a finite set of transitions. The set Φ consists of all Boolean combinations of atomic constraints of the form $r =$ or $r \neq$ for $r \in R$. The set U is the set of all functions from the set X of data word variables to $((\Gamma \times \hat{R}) \cup X)^*$.

It is required that

- For every $q \in Q$ and $x \in X$, there is at most one occurrence of x in $O(q)$ and
- for every transition $(q, \sigma, \phi, q', R', ud)$ and for every $x \in X$, x appears at most once in the set $\{ud(y) \mid y \in X\}$.

We say that the last two conditions above enforce a SSRT to be *copyless*, since it prevents multiple copies of contents being made.

A *valuation* val for a transducer S is a partial function over registers and data word variables such that for every register $r \in R$, either $val(r)$ is undefined or is a data value in D , and for every data word variable $x \in X$, $val(x)$ is a data word with origin information over Γ . The valuation val and data value d satisfies the atomic constraint $r =$ (resp. $r \neq$) if $val(r)$ is defined and $d = val(r)$ (resp. undefined or $d \neq val(r)$). Satisfaction is extended to Boolean combinations in the standard way. We say that a SSRT is *deterministic* if for every two transitions $(q, \sigma, \phi, q', R', u)$ and $(q, \sigma, \phi', q'', R'', u')$ with the same source state q and input symbol σ , the formulas ϕ and ϕ' are mutually exclusive (i.e., $\phi \wedge \phi'$ is unsatisfiable).

A configuration is a triple (q, val, i) where $q \in Q$ is a state, val is a valuation and i is the number of symbols read so far. The transducer starts in the configuration $(q_0, val_\epsilon, 0)$ where q_0 is the initial state and val_ϵ is the valuation such that $val_\epsilon(r)$ is undefined for every register $r \in R$ and $val_\epsilon(x) = \epsilon$ for every data word variable $x \in X$. From a configuration (q, val, i) , the transducer can read a pair $(\sigma, d) \in \Sigma \times D$ and go to the configuration $(q', val', i + 1)$ if there is a transition $(q, \sigma, \phi, q', R', ud)$ and 1) d and val satisfies ϕ and 2) val' is obtained from val by assigning d to all the registers in R' and for every $x \in X$, setting $val'(x)$ to $ud(x)[y \mapsto val(y), (\gamma, \text{curr}) \mapsto (\gamma, d, i + 1), (\gamma, r) \mapsto (\gamma, val(r), i + 1)]$ (this is obtained from $ud(x)$ by replacing every occurrence of y by $val(y)$ for every data word variable $y \in X$, replacing every occurrence of (γ, curr) by $(\gamma, d, i + 1)$ for every output

letter $\gamma \in \Gamma$ and replacing every occurrence of (γ, r) by $(\gamma, \text{val}(r), i + 1)$ for every output letter $\gamma \in \Gamma$ and every register $r \in R$). After reading a data word w , if the transducer reaches some configuration (q, val, n) and $O(q)$ is not defined, then the transducer's output $\llbracket S \rrbracket(w)$ is undefined for the input w . Otherwise, the transducer's output is defined as $\llbracket S \rrbracket(w) = O(q)[y \mapsto \text{val}(y), (\gamma, \text{curr}) \mapsto (\gamma, d, n), (\gamma, r) \mapsto (\gamma, \text{val}(r), n)]$, where d is the last data value in w .

Intuitively, the transition $(q, \sigma, \phi, q', R', ud)$ checks that the current valuation val and the data value d being read satisfies ϕ , goes to the state q' , stores d into the registers in R' and updates data word variables according to the update function ud . The condition that x appears at most once in the set $\{ud(y) \mid y \in X\}$ ensures that the contents of any data word variable are not duplicated into more than one variable. This ensures, among other things, that the length of the output is linear in the length of the input. The condition that for every two transitions $(q, \sigma, \phi, q', R', ud)$ and $(q, \sigma, \phi', q'', R'', ud')$ with the same source state and input symbol, the formulas ϕ and ϕ' are mutually exclusive ensures that the transducer cannot reach multiple configurations after reading a data word (i.e., the transducer is deterministic).

► **Example 2.** Consider the transduction that is identity on inputs in which the first and last data values are equal. On the remaining inputs, the output is the reverse of the input. This can be implemented by a SSRT using two data word variables. As each input symbol is read, it is appended to the front of the first variable and to the back of the second variable. The first variable stores the input and the second one stores the reverse. At the end, either the first or the second variable is output, depending on whether the last data value is equal or unequal to the first data value.

In Section 3, we define equivalence relations among data words and state our main result in terms of the finiteness of the indices of the equivalence relations. In Section 4, we prove that transductions satisfying certain properties can be implemented by SSRTs (the backward direction of the main result) and we prove the converse in Section 5.

3 How Prefixes and Suffixes Influence Each Other

First we adapt the concept of factored outputs introduced in [4] to data words. It is useful for observing parts of the output of a transduction that originate from some designated positions in the input while ignoring the rest.

► **Definition 3** (Factored outputs [4]). *Suppose f is a transduction and uvw is a data word over Σ . For a triple (γ, d, o) in $f(uvw)$, the abstract origin $\text{abs}(o)$ of o is **left** (resp. **middle**, **right**) if o is in u (resp. v , w). The factored output $f(\underline{u} \mid v \mid w)$ is obtained from $f(uvw)$ by first replacing every triple (γ, d, o) by $(*, *, \text{abs}(o))$ if $\text{abs}(o) = \text{left}$ (the other triples are retained without change). Then all consecutive occurrences of $(*, *, \text{left})$ are replaced by a single triple $(*, *, \text{left})$ to get $f(\underline{u} \mid v \mid w)$. Similarly we get $f(u \mid \underline{v} \mid w)$ and $f(u \mid v \mid \underline{w})$ by using $(*, *, \text{middle})$ and $(*, *, \text{right})$ respectively. We get $f(\underline{u} \mid v)$ and $f(u \mid \underline{v})$ similarly, except that there is no middle part.*

In $f(\underline{u} \mid v \mid w)$, the underline for u indicates that output positions whose origin is in u should be ignored. For $w = (a, d_1)(a, d_2)(b, d_3)(c, d_4)$ and the transduction f in Example 2, $f(w) = (c, d_4, 4)(b, d_3, 3)(a, d_2, 2)(a, d_1, 1)$ (assuming $d_4 \neq d_1$). The factored output $f((\underline{a}, d_1)(\underline{a}, d_2) \mid (b, d_3) \mid (c, d_4))$ is $(c, d_4, 4)(b, d_3, 3)(*, *, \text{left})$. Any device implementing this transduction will have to remember the first data value, since its equality/disequality with the last data value determines the output. The following definition formalizes this.

► **Definition 4** (Influencing values). Suppose f is a transduction. A data value d is f -suffix influencing in a data word u if there exists a data word v and a safe replacement d' for d in u such that $f(u[d/d'] | v) \neq f(u | v)$. A data value d is f -prefix influencing in a data word u if there exist data words u', v and a data value d' such that d does not occur in u' , d' is a safe replacement for d in $u \cdot u' \cdot v$ and $f(u \cdot u' | v[d/d']) \neq f(u \cdot u' | v)$. A data value d is f -influencing in a data word u if d is either f -suffix influencing or f -prefix influencing in u .

A simpler version of the above definition is given in [3] to identify data values that need to be remembered to recognize languages of data words. In the following illustrations, we use only one letter from the finite alphabet, so we ignore it in data words. Consider the transduction f of Example 2; let $u = d_1 d_2 d_3$, $v = d_1$ be data words and d'_1 be a data value such that d_1, d_2, d_3, d'_1 are all pairwise distinct. We have $f(u | v) = (*, *, \text{left})(d_1, 4)$ and $f(u[d_1/d'_1] | v) = (d_1, 4)(*, *, \text{left})$. Hence, d_1 is f -suffix influencing in u , according to Definition 4. For every data word v' and data value d'_2 that is a safe replacement for d_2 in u , $f(u | v) = f(u[d_2/d'_2] | v)$, so d_2 is not f -suffix influencing in u .

We illustrate prefix influencing values with a different example. Consider the transduction f whose output is ϵ for inputs of length less than five. For other inputs, the output is the third (resp. fourth) data value if the first and fifth are equal (resp. unequal). For the data word $u = d_1 d_2 d_3 d_4$, $f(u | \epsilon) = \epsilon$ and $f(u | v) = (*, *, \text{left}) = f(u[d_1/d'_1] | v)$ for all data values d'_1 and data words $v \neq \epsilon$. Hence, d_1 is not f -suffix influencing in u . However, any device implementing f must remember d_1 after reading u , so that it can be compared to the fifth data value. To capture data values like d_1 in u which are not suffix influencing but still need to be remembered, we have prefix influencing values in Definition 4. In u , d_1 is f -prefix influencing, since $f(d_1 d_2 d_3 d_4 | \underline{d_1}) = (d_3, 3) \neq (d_4, 4) = f(d_1 d_2 d_3 d_4 | \underline{d_5})$ (assuming d_1, \dots, d_5 are all pairwise distinct). In the data word $d_1 d_2$, d_1 is again f -prefix influencing. Note that $f(d_1 d_2 | \underline{v}) = \epsilon$ if $|v| < 3$ and $f(d_1 d_2 | \underline{v}) = (*, *, \text{right})$ otherwise. Hence, $f(d_1 d_2 | \underline{v})$ and $f(d_1 d_2 | \underline{v[d_1/d'_1]})$ will be equal for any data word v and data value d'_1 ; this will hide the influence of d_1 on the transformation. To observe the influence of d_1 , we first need to append some $u' = d_3 d_4$ to $u = d_1 d_2$ and ensure that the appended data word is not abstracted out (i.e., consider $f(d_1 d_2 d_3 d_4 | \underline{v})$ and not $f(d_1 d_2 | \underline{d_3 d_4 \cdot v})$) in the factored output used to define prefix influencing values. This is the role played by u' in Definition 4.

Suppose d_2, d_1 are two data values in some data word u . We say that d_1 occurs later than d_2 in u if the last occurrence of d_1 in u is to the right of the last occurrence of d_2 in u .

► **Definition 5.** Suppose f is a transduction and u is a data word. We denote by $\text{ifl}_f(u)$ the sequence $d_m \cdots d_1$, where $\{d_m, \dots, d_1\}$ is the set of all f -influencing values in u and for all $i \in [1, m]$, $(i-1)$ data values in $\{d_m, \dots, d_1\}$ occur later than d_i in u . We call d_i the i^{th} f -influencing data value in u . If a data value d is both f -prefix and f -suffix influencing in u , we say that d is of type **ps**. If d is f -suffix influencing but not f -prefix influencing (resp. f -prefix influencing but not f -suffix influencing) in u , we say that d is of type **s** (resp. **p**). We denote by $\text{aifl}_f(u)$ the sequence $(d_m, t(d_m)) \cdots (d_1, t(d_1))$, where $t(d_i)$ is the type of d_i for all $i \in [1, m]$.

► **Example 6.** Consider the transduction f defined as $f(u) = f_1(u) \cdot f_2(u)$; for $i \in [1, 2]$, f_i reverses its input if the i^{th} and last data values are distinct. On Other inputs, f_i is the identity (f_1 is the transduction given in Example 2). For $u = d_1 d_2 d_3 d_2 d_1$, d_1, d_2 are both f -suffix and f -prefix influencing; $\text{ifl}_f(u) = d_2 d_1$ and $\text{aifl}_f(u) = (d_2, \text{ps})(d_1, \text{ps})$.

Now we define relations among data words to identify those that behave similarly. For technical convenience, we assume that transductions are defined on all inputs. This reduces

some tediousness. Our results also hold without this assumption. Suppose f is a transduction, z is an integer and u, v are data words. We get $f_z(\underline{u} \mid v)$ from $f(\underline{u} \mid v)$ by replacing every triple (γ, d, o) by $(\gamma, d, o + z)$ (triples of the form $(*, *, \text{left})$ are not modified).

► **Definition 7.** For a transduction f , we define the relation \equiv_f on data words as $u_1 \equiv_f u_2$ if there exists a permutation π satisfying the following conditions, where $z = |u_1| - |u_2|$:

- $\lambda v.f_z(\pi(u_2) \mid v) = \lambda v.f(u_1 \mid v)$,
- $\text{aifl}_f(\pi(u_2)) = \text{aifl}_f(u_1)$ and
- for all u, v_1, v_2 , $f(u_1 \cdot u \mid v_1) = f(u_1 \cdot u \mid v_2)$ iff $f(\pi(u_2) \cdot u \mid v_1) = f(\pi(u_2) \cdot u \mid v_2)$.

As in the standard lambda calculus notation, $\lambda v.f_z(\underline{u} \mid v)$ denotes the function that maps each input v to $f_z(\underline{u} \mid v)$. Suppose $L \subseteq \Sigma^*$ is a language and $\mathbf{1}_L$ is its characteristic function. For the classical machine independent characterization of regular languages, two words w_1, w_2 are related if $\lambda v.\mathbf{1}_L(w_1 \cdot v) = \lambda v.\mathbf{1}_L(w_2 \cdot v)$. In the first condition in Definition 7, we replace the characteristic function $\mathbf{1}_L$ by the transduction f . The third condition takes into account the fact that for transductions, a suffix can influence the way a prefix is transformed: if the suffixes v_1 and v_2 transform the prefix $u_1 \cdot u$ in different ways, they transform the prefix $\pi(u_2) \cdot u$ also in different ways. Isomorphic words are always related by \equiv_f : if $\pi(u_2) = u_1$ for some permutation π , then π satisfies all the conditions of Definition 7, so $u_1 \equiv_f u_2$.

► **Lemma 8.** If f is invariant under permutations, then \equiv_f is an equivalence relation.

Consider the transduction f in Example 2. All non-empty data words are equivalent under \equiv_f . For any two non-empty data words u_1 and u_2 , a permutation π mapping the first data value in u_2 to the first data value in u_1 demonstrates this. For a data word u , $[u]_f$ is the equivalence class of \equiv_f containing u . All data words in the same equivalence of \equiv_f will have the same number of f -influencing data values. Hence, if \equiv_f has finite index, then there is a bound such that in any data word, the number of f -influencing values is less than the bound.

In Definition 7, the influencing values of two prefixes are aligned by applying a permutation to one of them. Below, similar alignment is achieved among infinitely many prefixes.

► **Definition 9.** Let f be a transduction and Π be the set of all permutations on D . An equalizing scheme for f is a function $E : (\Sigma \times D)^* \rightarrow \Pi$ such that there exists a sequence $\delta_1 \delta_2 \dots$ of data values satisfying the following condition: for every data word u and every i , the i^{th} f -influencing data value of $E(u)(u)$ (if it exists) is δ_i .

Note that $E(u)(u)$ denotes the application of the permutation $E(u)$ to the data word u . Left parts that have been equalized like this will not have arbitrary influencing data values — they will be from the sequence $\delta_1 \delta_2 \dots$. For the transduction in Example 2, the first data value is the only influencing value in any data word. An equalizing scheme will map the first data value of all data words to δ_1 . This is the counterpart of a standard trick in register automata, recognizing that to solve some problems, it is enough to consider data words over some finite subset of data values. Equalizing schemes achieve a similar purpose.

► **Definition 10.** For a transduction f and equalizing scheme E , we define the relation \equiv_f^E on data words as $v_1 \equiv_f^E v_2$ if for every data word u , $f(E(u)(u) \mid v_1) = f(E(u)(u) \mid v_2)$.

It is routine to verify that \equiv_f^E is an equivalence relation. Intuitively, $v_1 \equiv_f^E v_2$ if v_1 and v_2 behave similarly as right parts on all left parts that have been equalized by E . Either of the relations \equiv_f, \equiv_f^E may have finite index while the other one has infinite index; see e.g., [4, Example 4]. If E_1 and E_2 are two distinct equalizing schemes for f , then in general $\equiv_f^{E_1}$ and $\equiv_f^{E_2}$ are different. However, what matters is the index of the equivalence relations.

► **Lemma 11.** *Suppose f is a transduction that is invariant under permutations and without data peeking and E_1, E_2 are equalizing schemes. Then $\equiv_f^{E_1}$ and $\equiv_f^{E_2}$ have the same index.*

Following is the main result of this paper.

► **Theorem 12.** *A transduction f is implemented by a SSRT iff f satisfies the following properties: 1) f is invariant under permutations, 2) f is without data peeking, 3) f has linear blowup, 4) \equiv_f has finite index and 5) there exists an equalizing scheme E for f such that \equiv_f^E has finite index.*

The classical Myhill-Nerode theorem for regular languages over finite alphabets has only the counterpart of condition 4 above. Conditions 1 and 2 are needed since we have infinite alphabets. Conditions 3 and 5 are needed since we are characterizing streaming transducers and not language acceptors.

We prove the reverse direction of Theorem 12 in Section 4. The forward direction is proved in Section 5.

4 Constructing a SSRT from a Transduction

SSRTs read their input from left to right. Our first task is to get SSRTs to identify influencing data values as they are read one by one.

► **Lemma 13.** *Let f be a transduction, u be a data word, $\sigma \in \Sigma$ and d, e be distinct data values. If d is not f -suffix influencing (resp. f -prefix influencing) in u , then d is not f -suffix influencing (resp. f -prefix influencing) in $u \cdot (\sigma, e)$.*

A data value d that is not f -influencing in u will not become f -influencing just because some (σ, e) is appended to u . Hence, SSRTs can safely forget non-influencing values — they will not become influencing in the future. The next result proves the right congruence of \equiv_f .

► **Lemma 14.** *Suppose f is a transduction that is invariant under permutations and without data peeking. Suppose u_1, u_2 are data words such that $u_1 \equiv_f u_2$, $\text{ifl}_f(u_1) = d_1^m d_1^{m-1} \dots d_1^1$ and $\text{ifl}_f(u_2) = d_2^m d_2^{m-1} \dots d_2^1$. Suppose $d_1^0 \notin \text{data}(\text{ifl}_f(u_1), *)$, $d_2^0 \notin \text{data}(\text{ifl}_f(u_2), *)$ and $\sigma \in \Sigma$. For all $i, j \in [0, m]$, the following are true:*

1. d_1^i is f -suffix influencing (resp. f -prefix influencing) in $u_1 \cdot (\sigma, d_1^j)$ iff d_2^i is f -suffix influencing (resp. f -prefix influencing) in $u_2 \cdot (\sigma, d_2^j)$.
2. $u_1 \cdot (\sigma, d_1^j) \equiv_f u_2 \cdot (\sigma, d_2^j)$.

If $u \equiv_f u'$, d (resp. d') is the i^{th} f -influencing value in u (resp. u') and d is f -influencing in $u \cdot (\sigma, d)$, then d' is f -influencing in $u' \cdot (\sigma, d')$. To identify whether a newly read data value is influencing, a SSRT only needs to remember the equivalence class and the influencing values of the data word read so far. This is the idea behind the following construction.

► **Construction 15.** *Suppose f is a transduction that is invariant under permutations, \equiv_f has finite index and E is an equalizing scheme. Let I be the maximum number of f -influencing data values in any data word and $\delta_1 \dots \delta_I \in D^*$ be such that for any data word u , δ_i is the i^{th} f -influencing value in $E(u)(u)$. Consider a SSRT with the set of registers $R = \{r_1, \dots, r_I\}$. The states are of the form $([u]_f, \text{ptr})$, where u is some data word and $\text{ptr} : [1, |\text{ifl}_f(u)|] \rightarrow R$ is a pointer function. If $|\text{ifl}_f(u)| = 0$, then $\text{ptr} = \text{ptr}_\perp$, the trivial function from \emptyset to R . We let the set X of data word variables to be empty. Let ud_\perp be the trivial update function for the empty set X . The initial state is $([\epsilon]_f, \text{ptr}_\perp)$. Let δ_0 be an arbitrary data value in $D \setminus \{\delta_1, \dots, \delta_I\}$. From a state $([u]_f, \text{ptr})$, for every $\sigma \in \Sigma$ and*

$i \in [0, |\text{ifl}_f(u)|]$, there is a transition $(([u]_f, \text{ptr}), \sigma, \phi, ([E(u)(u) \cdot (\sigma, \delta_i)]_f, \text{ptr}'), R', \text{ud}_\perp)$. The condition ϕ is as follows, where $m = |\text{ifl}_f(u)|$: $\phi = \bigwedge_{j=1}^m \text{ptr}(j) \neq$ if $i = 0$ and $\phi = \text{ptr}(i) = \bigwedge_{j \in [1, m] \setminus \{i\}} \text{ptr}(j) \neq$ if $i \neq 0$. For every $j \in [1, |\text{ifl}_f(E(u)(u) \cdot (\sigma, \delta_i))|]$, $\text{ptr}'(j)$ is as follows: if the j^{th} f -influencing value of $E(u)(u) \cdot (\sigma, \delta_i)$ is the k^{th} f -influencing value of $E(u)(u)$ for some k , then $\text{ptr}'(j) = \text{ptr}(k)$. Otherwise, $\text{ptr}'(j) = r_{\text{reuse}} = \min(R \setminus \{\text{ptr}(k) \mid 1 \leq k \leq m, \delta_k \text{ is } f\text{-influencing in } E(u)(u) \cdot (\sigma, \delta_i)\})$ (minimum is based on the order $r_1 < r_2 < \dots < r_I$). The set R' is $\{r_{\text{reuse}}\}$ if $i = 0$ and δ_0 is f -influencing in $E(u)(u) \cdot (\sigma, \delta_0)$; R' is \emptyset otherwise.

Since all data words in the same equivalence class of f have the same number of f -influencing values and \equiv_f has finite index, I is finite in the above construction. It is routine to verify that the SSRT constructed above is deterministic. The definition of the next pointer function ptr' ensures that the register $\text{ptr}(j)$ always stores the j^{th} f -influencing value in the data word read so far. This is proved below using Lemma 13 and Lemma 14.

► **Lemma 16.** *Suppose the SSRT described in Construction 15 starts in the configuration $(([\epsilon]_f, \text{ptr}_\perp), \text{val}_\epsilon, 0)$ and reads some data word u . It reaches the configuration $(([u]_f, \text{ptr}), \text{val}, |u|)$ such that $\text{val}(\text{ptr}(i))$ is the i^{th} f -influencing value in u for all $i \in [1, |\text{ifl}_f(u)|]$.*

We extend the above construction to compute $\{f(E(u)(u) \mid \underline{v}) \mid v \in (\Sigma, D)^*\}$ where u is the data word read so far. It is enough to consider one representative v from every equivalence class of \equiv_f^E . If w is the entire input, the SSRT should output $f(w)$, which is same as $f(w \mid \underline{\epsilon}) = E(w)^{-1}(f(E(w)(w) \mid \underline{\epsilon}))$. The SSRT applies $E(u)^{-1}$ at every step instead of only at the end, so it computes $\{f(u \mid \underline{E(u)^{-1}(v)})\}$ after reading u . If $(\sigma, d) \in \Sigma \times D$ is the next symbol read, then $\{f(u \cdot (\sigma, d) \mid \underline{E(u \cdot (\sigma, d))^{-1}(v)})\}$ needs to be computed from whatever was computed for u .

To explain how the above computation is done, we use some terminology. In factored outputs of the form $f(u \mid \underline{v})$, $f(\underline{u} \mid \underline{v})$, $f(\underline{u} \mid v \mid \underline{w})$ or $f(\underline{u} \mid \underline{v} \mid \underline{w})$, a triple is said to come from u if it has origin in u or it is the triple $(*, *, \text{left})$. A left block in such a factored output is a maximal infix of triples, all coming from the left part u . Similarly, a non-right block is a maximal infix of triples, none coming from the right part. Middle blocks are defined similarly. For the transduction f in Example 2, $f((a, d_1)(b, d_2)(c, d_3))$ is $(c, d_3, 3)(b, d_2, 2)(a, d_1, 1)$. In $f((a, d_1)(b, d_2) \mid \underline{(c, d_3)})$, $(b, d_2, 2)(a, d_1, 1)$ is a left block. In $f((a, d_1) \mid (b, d_2) \mid \underline{(c, d_3)})$, $(b, d_2, 2)$ is a middle block. In $f(\underline{(a, d_1)} \mid (b, d_2) \mid \underline{(c, d_3)})$, $(*, *, \text{middle})(*, *, \text{left})$ is a non-right block, consisting of one middle and one left block. SSRTs will keep left blocks in variables, so we need a bound on the number of blocks.

► **Lemma 17.** *Suppose f is a transduction that is invariant under permutations and has linear blow up and E is an equalizing scheme such that \equiv_f^E has finite index. There is a bound $B \in \mathbb{N}$ such that for all data words u, v , the number of left blocks in $f(u \mid \underline{v})$ is at most B .*

Proof. Suppose for the sake of contradiction that there is no such bound B . Then there is an infinite family of pairs of data words $(u_1, v_1), (u_2, v_2), \dots$ such that for all $i \geq 1$, $f(u_i \mid \underline{v_i})$ has at least i left blocks. Applying any permutation to $f(u_i \mid \underline{v_i})$ will not change the number of left blocks. From Lemma 39, we infer that for all $i \geq 1$, $f(E(u_i)(u_i) \mid \underline{E(v_i)(v_i)})$ has at least i left blocks. Since \equiv_f^E has finite index, there is at least one equivalence class of \equiv_f^E that contains $E(u_i)(v_i)$ for infinitely many i . Let v be a data word from this equivalence class. From the definition of \equiv_f^E (Definition 10), we infer that for infinitely many i , $f(E(u_i)(u_i) \mid \underline{v})$ has at least i left blocks. Hence, for infinitely many i , $f(E(u_i)(u_i) \mid v)$ has at least $(i - 1)$ right blocks. Triples in the right blocks have origin in v . Since the number of positions in v is bounded, this contradicts the hypothesis that f has linear blow up. ◀

The concretization of the i^{th} left block (resp. middle block) in $f(\underline{u} \mid \underline{v} \mid \underline{w})$ is defined to be the i^{th} left block in $f(u \mid \underline{vw})$ (resp. the i^{th} middle block in $f(\underline{u} \mid v \mid \underline{w})$). The concretization of the i^{th} non-right block in $f(\underline{u} \mid \underline{v} \mid \underline{w})$ is obtained by concatenating the concretizations of the left and middle blocks that occur in the i^{th} non-right block. The following is a direct consequence of the definitions.

► **Proposition 18.** *The i^{th} left block of $f(u \cdot (\sigma, d) \mid \underline{v})$ is the concretization of the i^{th} non-right block of $f(\underline{u} \mid (\sigma, d) \mid \underline{v})$.*

For the transduction f from Example 2, the first left block of $f((a, d_1)(b, d_2) \mid (c, d_3))$ is $(b, d_2, 2)(a, d_1, 1)$, which is the concretization of $(*, *, \text{middle})(*, *, \text{left})$, the first non-right block of $f((a, d_1) \mid (b, d_2) \mid (c, d_3))$. Now suppose a SSRT needs to compute $f(u \cdot (\sigma, d) \mid E(u \cdot (\sigma, d))^{-1}(v))$ from whatever was computed for u . Proposition 18 implies that this can be done using $f(u \mid (\sigma, d) \cdot E(u \cdot (\sigma, d))^{-1}(v))$ and $f(\underline{u} \mid (\sigma, d) \mid E(u \cdot (\sigma, d))^{-1}(v))$, which provide the concretizations of the non-right blocks of $f(\underline{u} \mid (\sigma, d) \mid E(u \cdot (\sigma, d))^{-1}(v))$. This idea of progressively computing factored outputs was introduced in [4] for streaming string transducers over finite alphabets. However, we need more work here, since we need to identify a data word v' such that $f(u \mid (\sigma, d) \cdot E(u \cdot (\sigma, d))^{-1}(v)) = f(u \mid E(u)^{-1}(v'))$ (the SSRT would have already computed $f(u \mid E(u)^{-1}(v'))$). The next definition and the following lemma achieve this.

► **Definition 19.** *Suppose $\text{ifl}_f(E(u)(u)) = \delta_m \cdots \delta_1$, $\delta_0 \in D \setminus \{\delta_m, \dots, \delta_1\}$, $\eta \in \{\delta_0, \dots, \delta_m\}$ and $\sigma \in \Sigma$. We say that a permutation π tracks influencing values on $E(u)(u) \cdot (\sigma, \eta)$ if $\pi(\delta_i)$ is the i^{th} f -influencing value in $E(u)(u) \cdot (\sigma, \eta)$ for all $i \in [1, |\text{ifl}_f(E(u)(u) \cdot (\sigma, \eta))|]$.*

Lemma 13 implies that for $i \geq 2$ in the above definition, $\pi(\delta_i) \in \{\delta_m, \dots, \delta_1\}$ and $\pi(\delta_1) \in \{\delta_m, \dots, \delta_0\}$. We can infer from Lemma 14 that if $u \equiv_f u'$ and π tracks influencing values on $E(u')(u') \cdot (\sigma, \eta)$, then it also tracks influencing values on $E(u)(u) \cdot (\sigma, \eta)$.

► **Lemma 20.** *Suppose f is a transduction that is invariant under permutations and without data peeking, u, u', v are data words, $\sigma \in \Sigma$, $\text{ifl}_f(u) = d_m \cdots d_1$, $d_0 \in D \setminus \{d_m, \dots, d_1\}$, $\delta_0 \in D \setminus \{\delta_m, \dots, \delta_1\}$, $(d, \eta) \in \{(d_i, \delta_i) \mid i \in [0, m]\}$, π tracks influencing values on $E(u)(u) \cdot (\sigma, \eta)$ and $u \equiv_f u'$. Then $f(u \mid (\sigma, d) \cdot E(u \cdot (\sigma, d))^{-1}(v)) = f(u \mid E(u)^{-1}((\sigma, \eta) \cdot \pi(v)))$. If $(d, \eta) \in \{(d_i, \delta_i) \mid i \in [1, m]\}$, then $f(\underline{u} \mid (\sigma, d) \mid E(u \cdot (\sigma, d))^{-1}(v)) = E(u)^{-1}(f_z(E(u')(u') \mid (\sigma, \eta) \mid \pi(v)))$, where $z = |u| - |u'|$. If $(d, \eta) = (d_0, \delta_0)$, then $f(\underline{u} \mid (\sigma, d) \mid E(u \cdot (\sigma, d))^{-1}(v)) = E(u)^{-1} \odot \pi'(f_z(E(u')(u') \mid (\sigma, \eta) \mid \pi(v)))$, where π' is the permutation that interchanges δ_0 and $E(u)(d)$ and doesn't change any other data value (\odot denotes composition of permutations).*

Now $f(u \cdot (\sigma, d) \mid E(u \cdot (\sigma, d))^{-1}(v_i))$ can be computed for various v_i by taking concretizations from $f(u \mid (\sigma, d) \cdot E(u \cdot (\sigma, d))^{-1}(v_i))$, which are equal to $f(u \mid E(u)^{-1}((\sigma, \eta) \cdot \pi(v_i)))$ by Lemma 20. If $(\sigma, \eta) \cdot \pi(v_i) \equiv_f^E v'$, then it can be shown that $f(u \mid E(u)^{-1}((\sigma, \eta) \cdot \pi(v_i))) = f(u \mid E(u)^{-1}(v'))$. The left blocks of $f(u \mid E(u)^{-1}(v'))$ can be stored in variables in a SSRT and used to supply concretizations, but there is a problem. For $i \neq j$, suppose $(\sigma, \eta) \cdot \pi(v_i) \equiv_f^E (\sigma, \eta) \cdot \pi(v_j) \equiv_f^E v'$. Then $f(u \mid E(u)^{-1}(v'))$ is needed to compute both $f(u \cdot (\sigma, d) \mid E(u \cdot (\sigma, d))^{-1}(v_i))$ and $f(u \cdot (\sigma, d) \mid E(u \cdot (\sigma, d))^{-1}(v_j))$. Using the same variable twice is not allowed in SSRT transitions, which have to be copyless. We solve this problem by not computing $f(u \cdot (\sigma, d) \mid E(u \cdot (\sigma, d))^{-1}(v_i))$ and $f(u \cdot (\sigma, d) \mid E(u \cdot (\sigma, d))^{-1}(v_j))$ immediately. The SSRT remembers that there is a multiple dependency and continues to read the input until all but one of the dependencies vanish. Dependencies vanish as the SSRT reads more input symbols and gathers information. The SSRT uses tree structures in its states to keep track of multiple dependencies.

For a transduction f , let B be the maximum of the bounds on the number of left blocks shown in Lemma 17 and the number of middle blocks in factored outputs of the form $f(\underline{u} \mid (\sigma, d) \mid \underline{v})$. Let $(\Sigma \times D)^* / \equiv_f^E$ be the set of equivalence classes of \equiv_f^E , let $\hat{X} = \{\langle \theta, i \rangle \mid \theta \in ((\Sigma \times D)^* / \equiv_f^E)^*, 1 \leq i \leq B^2 + B\}$ and for $\theta \in ((\Sigma \times D)^* / \equiv_f^E)^*$, let $X_\theta = \{\langle \theta, i \rangle \mid 1 \leq i \leq B^2 + B\}$. We denote by $\theta \leftarrow$ the sequence obtained from θ by removing the right most equivalence class. We use a set $\mathcal{P} = \{P_1, \dots, P_B\}$ of parent references in the following definition. We use a finite subset of \hat{X} as data word variables to construct SSRTs.

► **Definition 21.** Suppose f is a transduction and E is an equalizing scheme for f . A dependency tree T is a tuple $(\Theta, \text{pref}, \text{bl})$, where the set of nodes Θ is a prefix closed finite subset of $((\Sigma \times D)^* / \equiv_f^E)^*$ and pref, bl are labeling functions. The root is ϵ and if $\theta \in \Theta \setminus \{\epsilon\}$, its parent is $\theta \leftarrow$. The labeling functions are $\text{pref} : \Theta \rightarrow (\Sigma \times D)^* / \equiv_f^E$ and $\text{bl} : \Theta \times [1, B] \rightarrow (\hat{X} \cup \mathcal{P})^*$. We call $\text{bl}(\theta, i)$ a block description. The dependency tree is said to be reduced if the following conditions are satisfied:

- every sequence θ in Θ has length that is bounded by $|(\Sigma \times D)^* / \equiv_f^E| + 1$,
- pref labels all the leaves with a single equivalence class of \equiv_f^E ,
- for every equivalence class $[v]_f^E$, there is exactly one leaf θ such that the last equivalence class in θ is $[v]_f^E$,
- $\text{bl}(\theta, i) \in (X_\theta \cup \mathcal{P})^*$ and is of length at most $2B + 1$ for all $\theta \in \Theta$ and $i \in [1, B]$ and
- for all $\theta \in \Theta$, each element of $X_\theta \cup \mathcal{P}$ occurs at most once in $\{\text{bl}(\theta, i) \mid 1 \leq i \leq B\}$.

If \equiv_f and \equiv_f^E have finite indices, there are finitely many possible reduced dependency trees. Suppose $\theta = \theta' \cdot [v]_f^E$ is in Θ , $\text{pref}(\theta) = [u]_f^E$ and $\text{bl}(\theta, 1) = P_1 \langle \theta, 1 \rangle P_2$. The intended meaning is that there is a data word u' that has been read by a SSRT and $u' \equiv_f u$. The block description $\text{bl}(\theta, 1) = P_1 \langle \theta, 1 \rangle P_2$ is a template for assembling the first left block of $f(u' \mid \underline{E(u')^{-1}(v)})$ from smaller blocks: take the first left block in the parent node θ' (P_1 refers to the first left block of the factored output assembled in the parent node), append to it the contents of the data word variable $\langle \theta, 1 \rangle$, then append the second left block in the parent node θ' . Intuitively, if $u' = u'' \cdot (\sigma, d)$, then the first non-right block of $f(\underline{u''} \mid (\sigma, d) \mid \underline{E(u')^{-1}(v)})$ is $(*, *, \text{left})(*, *, \text{middle})(*, *, \text{left})$ and P_1 refers to the concretization of the first left block $(*, *, \text{left})$, $\langle \theta, 1 \rangle$ contains the concretization of the first middle block $(*, *, \text{middle})$ and so on. The first left block in the parent node θ' itself may consist of some parent references and the contents of some other data word variables. This “unrolling” is formalized below.

► **Definition 22.** Suppose T is a dependency tree with set of nodes Θ . The function $\text{ur} : \Theta \times (\hat{X} \cup \mathcal{P})^* \rightarrow \hat{X}^*$ is defined as follows. For $\theta \in \Theta$ and $\mu \in (\hat{X} \cup \mathcal{P})^*$, $\text{ur}(\theta, \mu)$ is obtained from μ by replacing every occurrence of a parent reference P_i by $\text{ur}(\theta \leftarrow, \text{bl}(\theta \leftarrow, i))$ (replace $\text{bl}(\theta \leftarrow, i)$ by ϵ if $\theta \leftarrow = \epsilon$) for all i .

Intuitively, an occurrence of P_i in μ refers to the i^{th} left block in the parent node. If the current node is θ , the parent node is $\theta \leftarrow$, so we unroll μ by inductively unrolling the i^{th} left block of θ 's parent, which is given by $\text{ur}(\theta \leftarrow, \text{bl}(\theta \leftarrow, i))$. We are interested in dependency trees that allow to compute all factored outputs of the form $f(u \mid \underline{E(u)^{-1}(v)})$ by unrolling appropriate leaves. For convenience, we assume that $f(\epsilon) = \epsilon$. Let $T_\perp = (\{\epsilon\}, \text{pref}_\epsilon, \text{bl}_\epsilon)$, where $\text{pref}_\epsilon(\epsilon) = [\epsilon]_f^E$ and $\text{bl}_\epsilon(\epsilon, i) = \epsilon$ for all $i \in [1, B]$.

► **Definition 23.** Suppose f is a transduction, val is a valuation assigning a data word to every element of \hat{X} and T is a dependency tree. The pair (T, val) is complete for a data word u if $u = \epsilon$ and $T = T_\perp$, or $u \neq \epsilon$ and the following conditions are satisfied: for every equivalence class $[v]_f^E$, there exists a leaf node $\theta = \theta' \cdot [v]_f^E$ such that $\text{pref}(\theta) = [u]_f^E$ and for every i , the i^{th} left block of $f(u \mid \underline{E(u)^{-1}(v)})$ is $\text{val}(\text{ur}(\theta, \text{bl}(\theta, i)))$.

We construct SSRTs that will have dependency trees in its states, which will be complete for the data word read so far. As more symbols of the input data word are read, the dependency tree and the valuation for \hat{X} are updated as defined next.

► **Definition 24.** Suppose f is a transduction, E is an equalizing scheme and T is either T_\perp or a reduced dependency tree in which pref labels all the leaves with $[u]_f$ for some data word u . Suppose $\text{ifl}_f(u) = d_m \cdots d_1$, $d_0 \in D \setminus \{d_m, \dots, d_1\}$, $\delta_0 \in D \setminus \{\delta_m, \dots, \delta_1\}$, $(d, \eta) \in \{(d_i, \delta_i) \mid i \in [0, m]\}$ and $\sigma \in \Sigma$. Let π be a permutation tracking influencing values on $E(u)(u) \cdot (\sigma, \eta)$ as defined in Definition 19. For every equivalence class $[v]_f^E$, there is a leaf node $\theta_v = \theta' \cdot [(\sigma, \eta) \cdot \pi(v)]_f^E$ (or $\theta_v = \epsilon$, the root of the trivial dependency tree in case $u = \epsilon$). Let u' be an arbitrary data word in the equivalence class $[u]_f$. The (σ, η) extension of T is defined to be the tree obtained from T as follows: for every equivalence class $[v]_f^E$, create a new leaf $\theta = \theta_v \cdot [v]_f^E$ (with θ_v as parent) and set $\text{pref}(\theta) = [E(u')(u') \cdot (\sigma, \eta)]_f^E$. For every $i \in [1, B]$, let z be the i^{th} non-right block in $f(\overline{E(u')(u')} \mid (\sigma, \eta) \mid \overline{\pi(v)})$ (z is a sequence of left and middle blocks). Let z' be obtained from z by replacing j^{th} left block with P_j and k^{th} middle block with $\langle \theta, k \rangle$ for all j, k . Set $\text{bl}(\theta, i)$ to be z' . If there are internal nodes (nodes that are neither leaves nor the root) of this extended tree which do not have any of the newly added leaves as descendants, remove such nodes. The resulting tree T' is the (σ, η) extension of T . Suppose val is a valuation for \hat{X} such that (T, val) is complete for u . The (σ, d) extension val' of val is defined to be the valuation obtained from val by setting $\text{val}'(\langle \theta, k \rangle)$ to be the k^{th} middle block of $f(\underline{u} \mid (\sigma, d) \mid \overline{E(u \cdot (\sigma, d))^{-1}(v)})$ for every newly added leaf $\theta = \theta_v \cdot [v]_f^E$ and every $k \in [1, B]$. For all other variables, val' coincides with val . We call (T', val') the (σ, d) extension of (T, val) .

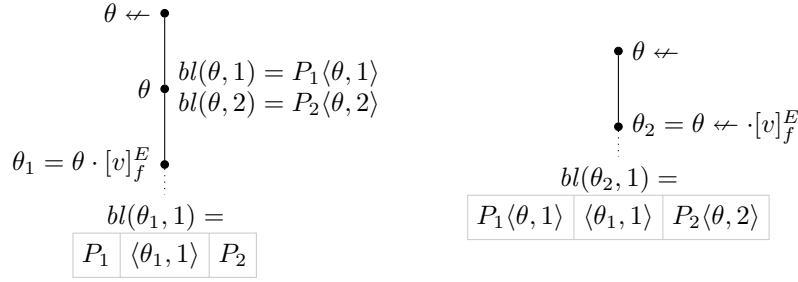
If some internal nodes are removed as described in Definition 24, it means that some dependencies have vanished due to the extension. For a newly added leaf θ , every element of $X_\theta \cup \mathcal{P}$ occurs at most once in $\{\text{bl}(\theta, i) \mid 1 \leq i \leq B\}$.

► **Lemma 25.** If (T, val) is complete for some data word u and (T', val') is the (σ, d) extension of (T, val) , then (T', val') is complete for $u \cdot (\sigma, d)$.

If (T, val) is complete for u and (T', val') is the (σ, d) extension of (T, val) , then the data word $\text{val}'(\langle \theta, k \rangle)$ is the k^{th} middle block of $f(\underline{u} \mid (\sigma, d) \mid \overline{E(u \cdot (\sigma, d))^{-1}(v)})$. We call $\langle \theta, k \rangle$ a *new middle block variable* and refer to it later for defining variable updates in transitions of SSRTs. The tree T' may not be reduced since it may contain branches that are too long. Next we see how to eliminate long branches.

► **Definition 26.** Suppose T is a dependency tree. A *shortening* of T is obtained from T as follows: let θ be an internal node that has only one child. Make the child of θ a child of θ 's parent, bypassing and removing the original node θ . Any descendent $\theta \cdot \theta'$ of θ in T is now identified by $\theta \leftarrow \cdot \theta'$. Set $\text{pref}(\theta \leftarrow \cdot \theta')$ to be $\text{pref}(\theta \cdot \theta')$, the label given by pref for the original descendent $\theta \cdot \theta'$ in T . Suppose $\theta \cdot [v]_f^E$ is the only child of θ in T . For every $i \in [1, B]$, set $\text{bl}(\theta \leftarrow \cdot [v]_f^E, i) = \mu$, where μ is obtained from $\text{bl}(\theta \cdot [v]_f^E, i)$ by replacing every occurrence of P_j by $\text{bl}(\theta, j)$. For strict descendants $\theta \leftarrow \cdot [v]_f^E \cdot \theta'$ of $\theta \leftarrow \cdot [v]_f^E$ and for every $i \in [1, B]$, set $\text{bl}(\theta \leftarrow \cdot [v]_f^E \cdot \theta', i) = \text{bl}(\theta \cdot [v]_f^E \cdot \theta', i)$.

Intuitively, θ has only one child, so only one factored output is dependent on the factored output stored in θ (all but one of the dependencies have vanished). Therefore, we can remove θ and pass on the information stored there to its only child. This is accomplished by replacing any occurrence of P_j in a block description of the child by $\text{bl}(\theta, j)$. Figure 1 shows an example, where θ_1 is the only child of θ . So θ is removed, θ_1 becomes θ_2 and a child of $\theta \leftarrow$.



■ **Figure 1** A dependency tree (left) and its shortening (right)

► **Lemma 27.** *If (T, val) is complete for a data word u and T' is a shortening of T , then (T', val) is also complete for u .*

Note that the valuation val need not be changed to maintain completeness of (T', val) . Hence, any new middle block variable will continue to store some middle block as before. Shortening will reduce the lengths of paths in the tree; still the resulting tree may not be reduced, since some node θ may have a block description $bl(\theta, i)$ that is too long and/or contains variables not in X_θ . Next we explain how to resolve this.

In a block description $bl(\theta, i)$, a non-parent block is any infix $bl(\theta, i)[j, k]$ such that 1) $j = 1$ or the $(j-1)^{\text{th}}$ element of $bl(\theta, i)$ is a parent reference, 2) $k = |bl(\theta, i)|$ or the $(k+1)^{\text{th}}$ element of $bl(\theta, i)$ is a parent reference and 3) for every $k' \in [j, k]$, the k'^{th} element of $bl(\theta, i)$ is not a parent reference. Intuitively, a non-parent block of $bl(\theta, i)$ is a maximal infix consisting of elements of \hat{X} only.

► **Definition 28.** *Suppose T is a dependency tree and val is a valuation for X . The trimming of T is obtained from T by performing the following for every node θ : enumerate the set $\{z \mid z \text{ is a non-parent block in } bl(\theta, i), 1 \leq i \leq B\}$ as z_1, z_2, \dots, z_r , choosing the order arbitrarily. If $bl(\theta, i)$ for some i contains z_j for some j , replace z_j by $\langle \theta, j \rangle$. Perform such replacements for all i and j . The trimming val' of val is obtained from val by setting $val'(\langle \theta, j \rangle) = val(z_j)$ for all j and $val'(\langle \theta', k \rangle) = \epsilon$ for all $\langle \theta', k \rangle$ occurring in any z_j . For elements of \hat{X} that neither occur in any z_j nor replace any z_j , val and val' coincide.*

For example, $bl(\theta_2, 1) = P_1 \langle \theta, 1 \rangle \langle \theta_1, 1 \rangle P_2 \langle \theta, 2 \rangle$ in Figure 1 is replaced by $P_1 \langle \theta_2, 1 \rangle P_2 \langle \theta_2, 2 \rangle$. In the new valuation, we have $val'(\langle \theta_2, 1 \rangle) = val(\langle \theta, 1 \rangle) \cdot val(\langle \theta_1, 1 \rangle)$, $val'(\langle \theta_2, 2 \rangle) = val(\theta, 2)$ and $val'(\langle \theta, 1 \rangle) = val'(\langle \theta_1, 1 \rangle) = val'(\langle \theta, 2 \rangle) = \epsilon$. The following result follows directly from definitions.

► **Proposition 29.** *If (T, val) is complete for a data word u , then so is the trimming (T', val') .*

States of the SSRT we construct will have reduced dependency trees. The following result is helpful in defining the SSRT transitions, where we have to say how to obtain a new tree from an old one.

► **Lemma 30.** *Suppose T is a reduced dependency tree or T_\perp , T_1 is the (σ, η) extension of T for some $(\sigma, \eta) \in \Sigma \times \{\delta_0, \delta_1, \dots\}$, T_2 is obtained from T_1 by shortening it as much as possible and T_3 is the trimming of T_2 . Then T_3 is a reduced dependency tree.*

Proof. Suppose all leaves in T are labeled with $[u]_f$ by *pref*. Then all leaves in T_1 (and hence in T_2 and T_3) are labeled by $[u \cdot (\sigma, \eta)]_f$. All paths in T_2 (and hence in T_3) are of length at

most $|(\Sigma \times D)^* / \equiv_f^E| + 1$: if there are longer paths, there will be at least $|(\Sigma \times D)^* / \equiv_f^E| + 1$ leaves since each internal node has at least two children. However, this is not possible since T_2 has only one leaf for every equivalence class of \equiv_f^E . In T_3 , for any node θ and any $i \in [1, B]$, $bl(\theta, i)$ will only contain elements from X_θ and \mathcal{P} , as ensured in the trimming process in Definition 28. There are at most B parent references, each of which occurs at most once in $bl(\theta, i)$ for at most one $i \in [1, B]$. Since every non-parent block is replaced by a data word variable in the trimming process, each $bl(\theta, i)$ is of length at most $2B + 1$. Each $bl(\theta, i)$ has at most $(B + 1)$ data word variables and $i \in [1, B]$, so at most $(B^2 + B)$ data word variables are sufficient for the block descriptions in θ . Hence, T_3 is reduced. \blacktriangleleft

We will now extend the SSRT constructed in Construction 15 to transform input data words to output data words with origin information. For any data word with origin information w , let $\downarrow_2(w)$ be the data word obtained from w by discarding the third component in every triple.

► **Construction 31.** Suppose f is a transduction satisfying all the conditions in Theorem 12. Let I be the maximum number of f -influencing values in any data word and let B be the maximum number of blocks in any factored output of the form $f(u \mid \underline{v})$ or $f(\underline{u} \mid v \mid \underline{w})$. Consider a SSRT with set of registers $R = \{R_1, \dots, R_I\}$ and data word variables $X = \{\langle \theta, i \rangle \mid \theta \in ((\Sigma \times D)^* / \equiv_f^E)^*, |\theta| \leq |(\Sigma \times D)^* / \equiv_f^E| + 1, i \in [1, B^2 + B]\}$. Every state is a triple $([u]_f, ptr, T)$ where u is some data word, T is a reduced dependency tree or T_\perp such that $pref$ labels every leaf in T with $[u]_f$ and $ptr : [1, |infl_f(u)|] \rightarrow R$ is a pointer function. The initial state is $([\epsilon]_f, ptr_\perp, T_\perp)$. Let $\delta_0 \notin \{\delta_{|infl_f(u)|}, \dots, \delta_1\}$ be an arbitrary data value. For every T and for every transition $(([u]_f, ptr), \sigma, \phi, ([E(u)(u) \cdot (\sigma, \delta_i)]_f, ptr'), R', ud_\perp)$ given in Construction 15, we will have the following transition: $(([u]_f, ptr, T), \sigma, \phi, ([E(u)(u) \cdot (\sigma, \delta_i)]_f, ptr', T'), R', ud)$. Let T_1 be the (σ, δ_i) extension of T and let T_2 be obtained from T_1 by shortening it as much as possible. T' is defined to be the trimming of T_2 . We define the update function ud using an intermediate function ud_1 and an arbitrary data word $u' \in [u]_f$. For every data word variable $\langle \theta, i \rangle$ that is not a new middle block variable in T_1 , set $ud_1(\langle \theta, i \rangle) = \langle \theta, i \rangle$. For every new middle block variable $\langle \theta, k \rangle$, say $\theta = \theta_v \cdot [v]_f^E$. Set $ud_1(\langle \theta, k \rangle) = \downarrow_2(z)$, where z is obtained from the k^{th} middle block of $f(E(u')(u') \mid (\sigma, \delta_i) \mid \pi(v))$ by replacing every occurrence of δ_j by $ptr(j)$ for all $j \in [1, |infl_f(u)|]$ and replacing every occurrence of δ_0 by $curr$. Here, π is a permutation tracking influencing values in $E(u')(u') \cdot (\sigma, \delta_i)$ as given in Definition 19. Next we define the function ud . While trimming T_2 , suppose a non-parent block z_j in a node θ was replaced by a data word variable $\langle \theta, j \rangle$. Define $ud(\langle \theta, j \rangle) = ud_1(z_j)$. For every data word variable $\langle \theta_1, k \rangle$ occurring in z_j , define $ud(\langle \theta_1, k \rangle) = \epsilon$. For all other data word variables $\langle \theta_2, k' \rangle$, define $ud(\langle \theta_2, k' \rangle) = ud_1(\langle \theta_2, k' \rangle)$. The output function O is defined as follows: for every state $([u]_f, ptr, T)$, $O([u]_f, ptr, T) = ur(\theta, bl(\theta, 1)) \dots ur(\theta, bl(\theta, B))$ where θ is the leaf of T such that $\theta = \theta' \cdot [\epsilon]_f^E$ ends in the equivalence class $[\epsilon]_f^E$.

Lemma 30 implies that if T is T_\perp or a reduced dependency tree, then so is T' . It is routine to verify that this SSRT is deterministic and copyless.

► **Lemma 32.** Let the SSRT constructed in Construction 31 be S . After reading a data word u , S reaches the configuration $(([u]_f, ptr, T), val, |u|)$ such that $ptr(i)$ is the i^{th} f -influencing value in u and (T, val) is complete for u .

Now we prove the reverse direction of our main result.

Proof of the reverse direction of Theorem 12. Let f be a transduction that satisfies all the properties stated in Theorem 12. Consider the SSRT constructed in Construction 31.

We infer from Lemma 32 that after reading any data word u , it will reach the configuration $(([u]_f, ptr, T), val, |u|)$ such that (T, val) is complete for u . The output function of the SSRT is such that $\llbracket S \rrbracket(u) = val(ur(\theta, bl(\theta, 1)) \cdots ur(\theta, bl(\theta, B)))$, where $\theta = \theta' \cdot [\epsilon]_f^E$ is the leaf of T ending with $[\epsilon]_f^E$. Since (T, val) is complete for u , we infer that $val(ur(\theta, bl(\theta, 1)) \cdots ur(\theta, bl(\theta, B)))$ is the concatenation of the left blocks of $f(u \mid \underline{E(u)^{-1}(\epsilon)}) = f(u)$. Hence, the SSRT S implements the transduction f . \blacktriangleleft

5 Properties of Transductions Implemented by SSRTs

In this section, we prove the forward direction of our main result (Theorem 12).

For a valuation val and permutation π , we denote by $\pi(val)$ the valuation that assigns $\pi(val(r))$ to every register r and $\pi(val(x))$ to every data word variable x . The following two results easily follow from definitions.

► **Proposition 33.** *Suppose a SSRT S reaches a configuration (q, val, n) after reading a data word u . If π is any permutation, then S reaches the configuration $(q, \pi(val), n)$ after reading $\pi(u)$.*

► **Proposition 34.** *If a SSRT S implements a transduction f , then f is invariant under permutations and is without data peeking.*

After a SSRT reads a data word, data values that are not stored in any of the registers will not influence the rest of the operations.

► **Lemma 35.** *Suppose a SSRT S implements the transduction f . Any data value d that is f -influencing in some data word u will be stored in one of the registers of S after reading u .*

Now we identify data words after reading which, a SSRT reaches similar configurations.

► **Definition 36.** *For a SSRT S , we define a binary relation \equiv_S on data words as follows: $u_1 \equiv_S u_2$ if they satisfy the following conditions. Suppose f is the transduction implemented by S , which reaches the configuration $(q_1, val_1, |u_1|)$ after reading u_1 and reaches $(q_2, val_2, |u_2|)$ after reading u_2 .*

1. $q_1 = q_2$,
2. for any two registers r_1, r_2 , we have $val_1(r_1) = val_1(r_2)$ iff $val_2(r_1) = val_2(r_2)$,
3. for any register r , $val_1(r)$ is the i^{th} f -suffix influencing value (resp. f -prefix influencing value) in u_1 iff $val_2(r)$ is the i^{th} f -suffix influencing value (resp. f -prefix influencing value) in u_2 ,
4. for any data word variable x , we have $val_1(x) = \epsilon$ iff $val_2(x) = \epsilon$ and
5. for any two subsets $X_1, X_2 \subseteq X$ and any arrangements χ_1, χ_2 of X_1, X_2 respectively, $val_1(\chi_1) = val_1(\chi_2)$ iff $val_2(\chi_1) = val_2(\chi_2)$.

An arrangement of a finite set X_1 is a sequence in X_1^* in which every element of X_1 occurs exactly once. It is routine to verify that \equiv_S is an equivalence relation of finite index.

The following result is shown by proving that \equiv_S refines \equiv_f .

► **Lemma 37.** *If a SSRT S implements a transduction f , then \equiv_f has finite index.*

Suppose a SSRT S reads a data word u , reaches the configuration $(q, val, |u|)$ and from there, continues to read a data word v . For some data word variable $x \in X$, if $val(x)$ is some data word z , then none of the transitions executed while reading v will split z — it might be appended or prepended with other data words and may be moved to other variables but never split. Suppose $X = \{x_1, \dots, x_m\}$. The transitions executed while reading v can

arrange $val(x_1), \dots, val(x_m)$ in various ways, possibly inserting other data words (whose origin is in v) in between. Hence, any left block of $\llbracket S \rrbracket(u \mid \underline{v})$ is $val(\chi)$, where χ is some arrangement of some subset $X' \subseteq X$.

► **Lemma 38.** *Suppose a SSRT S implements a transduction f . There is an equalizing scheme E for f such that \equiv_f^E has finite index.*

Proof of forward direction of Theorem 12. Suppose f is the transduction implemented by a SSRT S . Lemma 37 implies that \equiv_f has finite index and Lemma 38 implies that there exists an equalizing scheme E for f such that \equiv_f^E is of finite index. Proposition 34 implies that f is invariant under permutations and is without data peeking. The output of S on any input is the concatenation of the data words stored in some variables in S and constantly many symbols coming from the output function of S . The contents of data word variables are generated by transitions when reading input symbols and each transition can write only constantly many symbols into any data word variable after reading one input symbol. After some content is written into a data word variable, it is never duplicated into multiple copies since the transitions of S are copyless. Hence, any input position can be the origin of only constantly many output positions. Hence, f has linear blow up. ◀

6 Future Work

One direction to explore is whether there is a notion of minimal canonical SSRT and if a given SSRT can be reduced to an equivalent minimal one. Adding a linear order on the data domain, logical characterization of SSRTs and studying two way transducer models with data are some more interesting studies.

Using nominal automata, techniques for finite alphabets can often be elegantly carried over to infinite alphabets, as done in [14], for example. It would be interesting to see if the same can be done for streaming transducers over infinite alphabets.

References

- 1 R. Alur and P. Černý. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *POPL 2011*, POPL, pages 1–12. ACM, 2011.
- 2 D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.
- 3 M. Benedikt, C. Ley, and G. Puppis. What you must remember when processing data words. In *Proceedings of the 4th Alberto Mendelzon International Workshop on Foundations of Data Management, Argentina*, volume 619 of *CEUR Workshop Proceedings*, 2010.
- 4 M. Bojańczyk. Transducers with origin information. In *ICALP*, volume 8573 of *LNCS*, pages 26–37, Berlin, Heidelberg, 2014. Springer.
- 5 S. Cassel, F. Howar, B. Jonsson, M. Merten, and B. Steffen. A succinct canonical register automaton model. *Journal of Logical and Algebraic Methods in Programming*, 84(1):54 – 66, 2015. Special Issue: The 23rd Nordic Workshop on Programming Theory (NWPT 2011) Special Issue: Domains X, International workshop on Domain Theory and applications, Swansea, 5-7 September, 2011.
- 6 S. Cassel, B. Jonsson, F. Howar, and B. Steffen. A succinct canonical register automaton model for data domains with binary relations. In *Automated Technology for Verification and Analysis - 10th International Symposium, 2012, Proceedings*, pages 57–71, 2012. doi: 10.1007/978-3-642-33386-6_6.
- 7 Y-F Chen, O. Lengál, T. Tan, and Z. Wu. Register automata with linear arithmetic. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017.

- 8 M. Chytil and V. Jákł. Serial composition of 2-way finite-state transducers and simple programs on strings. In *Automata, Languages and Programming, Fourth Colloquium, University of Turku, Finland, July 18-22, 1977, Proceedings*, pages 135–147. Springer Berlin Heidelberg, 1977.
- 9 A. Durand-Gasselin and P. Habermehl. Regular transformations of data words through origin information. In B. Jacobs and C. Löding, editors, *Foundations of Software Science and Computation Structures*, pages 285–300, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.
- 10 N. Francez and M. Kaminski. An algebraic characterization of deterministic regular languages over infinite alphabets. *Theoretical Computer Science*, 306:155–175, 2003.
- 11 F. Howar, M. Isberner, B. Steffen, O. Bauer, and B. Jonsson. Inferring semantic interfaces of data structures. In T. Margaria and B. Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, pages 554–571, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- 12 F. Howar, B. Steffen, B. Jonsson, and S. Cassel. Inferring canonical register automata. In V. Kuncak and A. Rybalchenko, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 251–266, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- 13 M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329 – 363, 1994.
- 14 J. Moerman, M. Sammartino, A. Silva, B. Klin, and M. Szyrwelski. Learning nominal automata. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 613–625, 2017.

A

 Fundamental Properties of Transductions

The following result says that if a transduction is invariant under permutations, then so are all its factored outputs.

► **Lemma 39.** *Suppose f is a transduction that is invariant under permutations, u, v, w are data words, π is any permutation and z is any integer. Then $\pi(f_z(\underline{u} \mid v)) = f_z(\pi(\underline{u}) \mid \pi(v))$, $\pi(f_z(u \mid \underline{v})) = f_z(\pi(u) \mid \pi(\underline{v}))$ and $\pi(f_z(\underline{u} \mid v \mid \underline{w})) = f_z(\pi(\underline{u}) \mid \pi(v) \mid \pi(\underline{w}))$.*

Proof. From the invariance of f under permutations, we have $f(\pi(u) \cdot \pi(v)) = \pi(f(u \cdot v))$. Adding z to every triple on both sides, we get

$$f_z(\pi(u) \cdot \pi(v)) = \pi(f_z(u \cdot v)) .$$

For every $i \in [1, |f_z(\pi(u) \cdot \pi(v))|]$, we perform the following on the LHS of the above equation: let (γ, d, o) be the i^{th} triple in the LHS; if $o - z \in [1, |u|]$, replace the triple by $(*, *, \text{left})$. After performing this change for every i , merge consecutive occurrences of $(*, *, \text{left})$ into a single triple $(*, *, \text{left})$. At the end, we get $f_z(\pi(u) \mid \pi(v))$.

Now perform exactly the same operations not on the RHS $\pi(f_z(u \cdot v))$, but on $f_z(u \cdot v)$. The i^{th} triple will be $(\gamma, \pi^{-1}(d), o)$ and it changes to $(*, *, \text{left})$ iff the i^{th} triple (γ, d, o) in the LHS changed to $(*, *, \text{left})$. Now, if we merge consecutive occurrences of $(*, *, \text{left})$ into a single triple $(*, *, \text{left})$, we get $f_z(\underline{u} \mid v)$. If we now apply the permutation π to this, we get $\pi(f_z(\underline{u} \mid v))$, but we also get exactly the same sequence of triples we got from LHS after the changes, which is $f_z(\pi(\underline{u}) \mid \pi(v))$. Hence, $f_z(\pi(\underline{u}) \mid \pi(v)) = \pi(f_z(\underline{u} \mid v))$. The proofs of the other two equalities are similar. ◀

The following result says that the influencing values of a data word are affected by a permutation as expected.

► **Lemma 40.** *If f is a transduction that is invariant under permutations and u is a data word, then for any permutation π , $\text{aifl}_f(\pi(u)) = \pi(\text{aifl}_f(u))$.*

Proof. It is sufficient to prove that for any position j of u , the data value in the j^{th} position of u is a f -suffix influencing value in u iff the data value in the j^{th} position of $\pi(u)$ is a f -suffix influencing value in $\pi(u)$ and similarly for f -prefix influencing values. Indeed, suppose d is the data value in the j^{th} position of u and it is a f -suffix influencing value in u . By Definition 4, there exists a data word v and a data value d' that is a safe replacement for d in u such that $f(u[d/d'] \mid v) \neq f(\underline{u} \mid v)$. The data value at j^{th} position of $\pi(u)$ is $\pi(d)$ and the word $\pi(v)$ and the data value $\pi(d')$ witnesses that $\pi(d)$ is a f -suffix influencing in $\pi(u)$. Indeed, if $f(u[d/d'] \mid v) \neq f(\underline{u} \mid v)$, then Lemma 39 implies that $f(\pi(u)[\pi(d)/\pi(d')] \mid \pi(v)) \neq f(\pi(\underline{u}) \mid \pi(v))$. The converse direction of the proof is symmetric, using the permutation π^{-1} .

Suppose d is the data value in the j^{th} position of u and it is a f -prefix influencing value in u . By Definition 4, there exist data words u', v and a data value d' that is a safe replacement for d in $u \cdot u' \cdot v$ such that d doesn't occur in u' and $f(u \cdot u' \mid \underline{v}) \neq f(u \cdot u' \mid \underline{v[d/d']})$. The data value at j^{th} position of $\pi(u)$ is $\pi(d)$ and the words $\pi(u'), \pi(v)$ and the data value $\pi(d')$ witnesses that $\pi(d)$ is a f -prefix influencing in $\pi(u)$. Indeed, since $f(u \cdot u' \mid \underline{v}) \neq f(u \cdot u' \mid \underline{v[d/d']})$, Lemma 39 implies that $f(\pi(u) \cdot \pi(u') \mid \pi(\underline{v})) \neq f(\pi(u) \cdot \pi(u') \mid \pi(\underline{v}[\pi(d)/\pi(d')]))$. The converse direction of the proof is symmetric, using the permutation π^{-1} . ◀

A data value that does not occur in a data word can not influence how it is transformed.

► **Lemma 41.** *Suppose f is a transduction that is invariant under permutations and without data peeking and a data value d is f -prefix influencing in a data word u . Then d occurs in u .*

Proof. Suppose d does not occur in u . We will prove that d is not f -prefix influencing in u . Let u', v be any data words such that d does not occur in u' . Suppose d' is a safe replacement for d in $u \cdot u' \cdot v$. Let π be the permutation that interchanges d and d' and does not change any other value. Neither d nor d' occurs in $u \cdot u'$, so $\pi(u \cdot u') = u \cdot u'$. The data value d' does not occur in v , so $\pi(v) = v[d/d']$. Since f is without data peeking, only data values in occurring in $u \cdot u'$ occur $f(u \cdot u' \mid v)$, so neither d nor d' occur in $f(u \cdot u' \mid v)$, so $\pi(f(u \cdot u' \mid v)) = f(u \cdot u' \mid v)$. Since f is invariant under permutations, we infer from Lemma 39 that $\pi(f(u \cdot u' \mid v)) = f(\pi(u \cdot u') \mid \pi(v))$. This implies that $f(u \cdot u' \mid v) = f(u \cdot u' \mid v[d/d'])$. Hence, d is not f -prefix influencing in u . ◀

Data values in a prefix can be permuted without changing the way it affects suffixes, as long as we don't change the influencing values.

► **Lemma 42.** *Suppose f is a transduction that is invariant under permutations, u, v are data words and π is any permutation that is identity on the set of data values that are f -influencing in u . Then $f(\pi(u) \mid v) = f(u \mid v)$ and $\text{aifl}_f(u) = \text{aifl}_f(\pi(u))$.*

Proof. Let $\{d_1, \dots, d_n\}$ be the set of all data values occurring in u that are not f -influencing in u . Let d'_1, \dots, d'_n be safe replacements for d_1, \dots, d_n respectively in u , such that $\{d'_1, \dots, d'_n\} \cap (\{d_1, \dots, d_n\} \cup \{\pi(d_1), \dots, \pi(d_n)\}) = \emptyset$. Since d_1 is not f -suffix influencing in u , we have $f(u[d_1/d'_1] \mid v) = f(u \mid v)$. Since d_2 is not f -influencing in u , we infer from Lemma 51 that d_2 is not f -influencing in $u[d_1/d'_1]$. Hence, $f(u[d_1/d'_1][d_2/d'_2] \mid v) = f(u[d_1/d'_1] \mid v) = f(u \mid v)$. Also from Lemma 51, we infer that d'_1 is not f -influencing in $u[d_1/d'_1]$ (put $e = d'_1$ in Lemma 51 to see this). Similarly, neither d'_1 nor d'_2 are f -influencing in $u[d_1/d'_1][d_2/d'_2]$. On the other hand, we infer from Lemma 51 that all the data values that are f -suffix influencing (resp. f -prefix influencing) in u are also f -suffix influencing (resp. f -prefix influencing) in $u[d_1/d'_1][d_2/d'_2]$. This reasoning can be routinely extended to an induction on i to infer that $f(u[d_1/d'_1, \dots, d_i/d'_i] \mid v) = f(u \mid v)$ and d'_1, \dots, d'_i are not f -influencing in $u[d_1/d'_1, \dots, d_i/d'_i]$. Hence, $f(u[d_1/d'_1, \dots, d_n/d'_n] \mid v) = f(u \mid v)$. In addition, all the data values that are f -suffix influencing (resp. f -prefix influencing) in u are also f -suffix influencing (resp. f -prefix influencing) in $u[d_1/d'_1, \dots, d_n/d'_n]$.

Now we prove that $\pi(d_1), \dots, \pi(d_n)$ are safe replacements for d'_1, \dots, d'_n in $u[d_1/d'_1, \dots, d_n/d'_n]$. We know that $\text{data}(u[d_1/d'_1, \dots, d_n/d'_n], *) = \{d'_1, \dots, d'_n\} \cup \{d \mid d \text{ is } f\text{-influencing in } u\}$. We have $\{\pi(d_1), \dots, \pi(d_n)\} \cap \{d'_1, \dots, d'_n\} = \emptyset$ by choice. Since π is identity on $\{d \mid d \text{ is } f\text{-influencing in } u\}$ and d_1, \dots, d_n are not f -influencing in u , we have $\{\pi(d_1), \dots, \pi(d_n)\} \cap \{d \mid d \text{ is } f\text{-influencing in } u\} = \emptyset$. This proves that $\pi(d_1), \dots, \pi(d_n)$ are safe replacements for d'_1, \dots, d'_n in $u[d_1/d'_1, \dots, d_n/d'_n]$.

As we did in the first paragraph of this proof, we conclude that $f(u[d_1/d'_1, \dots, d_n/d'_n][d'_1/\pi(d_1), \dots, d'_n/\pi(d_n)] \mid v) = f(u[d_1/d'_1, \dots, d_n/d'_n] \mid v) = f(u \mid v)$. Since $u[d_1/d'_1, \dots, d_n/d'_n][d'_1/\pi(d_1), \dots, d'_n/\pi(d_n)] = u[d_1/\pi(d_1), \dots, d_n/\pi(d_n)] = \pi(u)$, we infer that $f(\pi(u) \mid v) = f(u \mid v)$. In addition, $\pi(d_1), \dots, \pi(d_n)$ are not f -influencing in $\pi(u)$ and all the values that are f -suffix influencing (resp. f -prefix influencing) in u are also f -suffix influencing (resp. f -prefix influencing) in $\pi(u)$. Hence, $\text{aifl}(\pi(u)) = \text{aifl}(u)$. ◀

Data values in a suffix can be permuted without changing the way it affects prefixes, as long as we don't change the prefix influencing values.

► **Lemma 43.** *Suppose f is a transduction that is invariant under permutations and without data peeking, u, v are data data words and π is any permutation that is identity on the set of data values that are f -prefix influencing in u . Then $f(u \mid \pi(v)) = f(u \mid v)$.*

Proof. Let $\{d_1, \dots, d_n\}$ be the set of all data values occurring in v that are not f -prefix influencing in u . Let d'_1, \dots, d'_n be safe replacements for d_1, \dots, d_n respectively in $u \cdot v$, such that $\{d'_1, \dots, d'_n\} \cap (\{d_1, \dots, d_n\} \cup \{\pi(d_1), \dots, \pi(d_n)\}) = \emptyset$. Since d_1 is not f -prefix influencing in u , we have $f(u \mid v[d_1/d'_1]) = f(u \mid v)$. Since d_2 is not f -prefix influencing in u , we have $f(u \mid v[d_1/d'_1][d_2/d'_2]) = f(u \mid v[d_1/d'_1]) = f(u \mid v)$. The same reasoning can be used in an induction to conclude that $f(u \mid v[d_1/d'_1, d_2/d'_2, \dots, d_n/d'_n]) = f(u \mid v)$.

Now we will prove that $\pi(d_1), \dots, \pi(d_n)$ are safe replacements for d'_1, \dots, d'_n respectively in $v[d_1/d'_1, d_2/d'_2, \dots, d_n/d'_n]$. We have $\text{data}(v[d_1/d'_1, \dots, d_n/d'_n], *) = \{d'_1, \dots, d'_n\} \cup \{d \mid d \text{ is } f\text{-prefix influencing in } u\}$. We have $\{\pi(d_1), \dots, \pi(d_n)\} \cap \{d'_1, \dots, d'_n\} = \emptyset$ by choice. Since π is identity on $\{d \mid d \text{ is } f\text{-prefix influencing in } u\}$ and d_1, \dots, d_n are not f -prefix influencing in u , we have $\{\pi(d_1), \dots, \pi(d_n)\} \cap \{d \mid d \text{ is } f\text{-prefix influencing in } u\} = \emptyset$. This proves that $\pi(d_1), \dots, \pi(d_n)$ are safe replacements for d'_1, \dots, d'_n in $v[d_1/d'_1, \dots, d_n/d'_n]$.

Now we claim that $f(u \mid v[d_1/d'_1, d_2/d'_2, \dots, d_n/d'_n][d'_1/\pi(d_1)]) = f(u \mid v[d_1/d'_1, d_2/d'_2, \dots, d_n/d'_n])$. Suppose not, i.e., $f(u \mid v[d_1/d'_1, d_2/d'_2, \dots, d_n/d'_n][d'_1/\pi(d_1)]) \neq f(u \mid v[d_1/d'_1, d_2/d'_2, \dots, d_n/d'_n])$. This can be written equivalently as $f(u \mid v[d_1/d'_1, d_2/d'_2, \dots, d_n/d'_n][d'_1/\pi(d_1)]) \neq f(u \mid v[d_1/d'_1, d_2/d'_2, \dots, d_n/d'_n][d'_1/\pi(d_1)][\pi(d_1)/d'_1])$. Then we infer from Definition 4 that $\pi(d_1)$ is f -prefix influencing in u , which contradicts the hypothesis that π is identity on all values that are f -prefix influencing in u . Hence, $f(u \mid v[d_1/d'_1, d_2/d'_2, \dots, d_n/d'_n][d'_1/\pi(d_1)]) = f(u \mid v[d_1/d'_1, d_2/d'_2, \dots, d_n/d'_n])$.

Similar reasoning can then be used to infer that $f(u \mid v[d_1/d'_1, d_2/d'_2, \dots, d_n/d'_n][d'_1/\pi(d_1), \dots, d'_n/\pi(d_n)]) = f(u \mid v[d_1/d'_1, d_2/d'_2, \dots, d_n/d'_n]) = f(u \mid v)$. Hence, $f(u \mid \pi(v)) = f(u \mid v)$. ◀

If two factored outputs are equal, factoring out the same word from the same positions of the inputs will not destroy the equality.

► **Lemma 44.** *Suppose f is a transduction, u, u_1, u_2, v, v_1, v_2 are data words, $\sigma \in \Sigma$, d is a data value and $z = |u_1| - |u_2|$.*

1. *If $f(u_1 \mid u \cdot v) = f_z(u_2 \mid u \cdot v)$, then $f(u_1 \mid u \mid v) = f_z(u_2 \mid u \mid v)$.*
2. *If $f(u_1 \mid u \cdot v) = f_z(u_2 \mid u \cdot v)$, then $f(u_1 \cdot u \mid v) = f_z(u_2 \cdot u \mid v)$.*
3. *If $f(u \cdot v \mid v_1) = f(u \cdot v \mid v_2)$, then $f(u \mid v \cdot v_1) = f(u \mid v \cdot v_2)$.*
4. *If $f(u \cdot v \mid v_1) = f(u \cdot v \mid v_2)$, then $f(u \mid v \mid v_1) = f(u \mid v \mid v_2)$.*

Proof. We prove the first statement. Others are similar. We have the following equality from the hypothesis.

$$f(u_1 \mid u \cdot v) = f_z(u_2 \mid u \cdot v)$$

For every $i \in [1, |f(u_1 \mid u \cdot v)|]$, we perform the following on the LHS of the above equation: let (γ, d, o) be the i^{th} triple in the LHS; if $o > |u_1| + |u|$, replace the triple by $(*, *, \text{right})$ (the origin of such a triple is in v). Otherwise, don't change the triple. After performing this change for every i , merge consecutive occurrences of $(*, *, \text{right})$ into a single triple $(*, *, \text{right})$. At the end, we get $f(u_1 \mid u \mid v)$.

Now perform exactly the same operations on the RHS $f_z(u_2 \mid u \cdot v)$. The i^{th} triple (γ, d, o) will change to $(*, *, \text{right})$ (resp. will not change) iff the i^{th} triple (γ, d, o) in the LHS changed to $(*, *, \text{right})$ (resp. did not change). Note that if $o > |u_1| + |u|$, $o - z > |u_2| + |u|$. Hence, the triples that change to $(*, *, \text{right})$ in the RHS are precisely the triples whose origin is in v . Now, if we merge consecutive occurrences of $(*, *, \text{right})$ into a single triple

$(*, *, \text{right})$, we get $f_z(\underline{u}_2 \mid u \mid \underline{v})$. This is also the same sequence of triples we got from LHS after the changes, which is $f(\underline{u}_1 \mid u \mid \underline{v})$. Hence, $f(\underline{u}_1 \mid u \mid \underline{v}) = f_z(\underline{u}_2 \mid u \mid \underline{v})$. ◀

► **Lemma 45.** *Suppose f is a transduction that is invariant under permutations, u, v, w are data words and $\pi, \pi' \in \Pi$ are permutations on the data domain D . If π and π' coincide on those data values that are f -influencing in $u \cdot v$, then $\pi(f(\underline{u} \mid v \mid \underline{w})) = f(\pi(\underline{u}) \mid \pi(v) \mid \pi'(w))$.*

Proof. Since π and π' coincide on those data values that are f -influencing in $u \cdot v$, we infer from Lemma 43 that $f(\pi(u \cdot v) \mid \pi(w)) = f(\pi(u \cdot v) \mid \pi'(w))$. From point 4 of Lemma 44, we conclude that $f(\pi(\underline{u}) \mid \pi(v) \mid \pi(w)) = f(\pi(\underline{u}) \mid \pi(v) \mid \pi'(w))$. We have from Lemma 39 that $\pi(f(\underline{u} \mid v \mid \underline{w})) = f(\pi(\underline{u}) \mid \pi(v) \mid \pi(w))$. Combining the last two equalities, we get the result. ◀

The following result is in some sense the converse of points (3) and (4) in Lemma 44.

► **Lemma 46.** *Let f be a transduction and u, v, w_1, w_2 be data words. If $f(\underline{u} \mid v \mid \underline{w}_1) = f(\underline{u} \mid v \mid \underline{w}_2)$ and $f(u \mid \underline{vw}_1) = f(u \mid \underline{vw}_2)$, then $f(uv \mid \underline{w}_1) = f(uv \mid \underline{w}_2)$.*

Proof. The number of occurrences of the triple $(*, *, \text{right})$ is the same in $f(\underline{u} \mid v \mid \underline{w}_1)$ and $f(uv \mid \underline{w}_1)$. The number of occurrences of the triple $(*, *, \text{right})$ is the same in $f(\underline{u} \mid v \mid \underline{w}_2)$ and $f(uv \mid \underline{w}_2)$. Suppose $f(uv \mid \underline{w}_1) \neq f(uv \mid \underline{w}_2)$. If the number of occurrences of the triple $(*, *, \text{right})$ are different in $f(uv \mid \underline{w}_1)$ and $f(uv \mid \underline{w}_2)$, then the number of occurrences of the triple $(*, *, \text{right})$ are different in $f(\underline{u} \mid v \mid \underline{w}_1)$ and $f(\underline{u} \mid v \mid \underline{w}_2)$ and we are done. So assume that the number of occurrences of the triple $(*, *, \text{right})$ is the same in $f(uv \mid \underline{w}_1)$ and $f(uv \mid \underline{w}_2)$. Let i be the first position where $f(uv \mid \underline{w}_1)$ and $f(uv \mid \underline{w}_2)$ differ.

Case 1: at position i , $f(uv \mid \underline{w}_1)$ contains $(*, *, \text{right})$ and $f(uv \mid \underline{w}_2)$ contains a triple whose origin is in u or v . If the i^{th} triple in $f(uv \mid \underline{w}_2)$ has origin in u , there will be a position in $f(u \mid \underline{vw}_2)$ that will have a triple whose origin is in u and the same position in $f(u \mid \underline{vw}_1)$ will have $(*, *, \text{right})$ and we are done. If the i^{th} triple in $f(uv \mid \underline{w}_2)$ has origin in v , there will be a position in $f(\underline{u} \mid v \mid \underline{w}_2)$ that will have a triple whose origin is in v and the same position in $f(\underline{u} \mid v \mid \underline{w}_1)$ will have $(*, *, \text{right})$ and we are done.

Case 2: at position i , $f(uv \mid \underline{w}_2)$ contains $(*, *, \text{right})$ and $f(uv \mid \underline{w}_1)$ contains a triple whose origin is in u or v . This can be handled similarly as above, with the role of w_1 and w_2 interchanged.

Case 3: at position i , $f(uv \mid \underline{w}_1)$ contains a triple whose origin is in u and $f(uv \mid \underline{w}_2)$ contains a triple whose origin is in v . In this case, $f(\underline{u} \mid v \mid \underline{w}_1)$ will have a position with the triple $(*, *, \text{left})$ and the same position in $f(\underline{u} \mid v \mid \underline{w}_2)$ will have a triple whose origin is in v and we are done.

Case 4: at position i , $f(uv \mid \underline{w}_1)$ contains a triple whose origin is in v and $f(uv \mid \underline{w}_2)$ contains a triple whose origin is in u . This case can be handled similarly as above.

Case 5: at position i , both $f(uv \mid \underline{w}_1)$ and $f(uv \mid \underline{w}_2)$ has triples whose origin is in u but the contents are different. In this case, there will be a position where $f(u \mid \underline{vw}_1)$ and $f(u \mid \underline{vw}_2)$ differ and we are done.

Case 6: at position i , both $f(uv \mid \underline{w}_1)$ and $f(uv \mid \underline{w}_2)$ has triples whose origin is in v but the contents are different. In this case, there will be a position where $f(\underline{u} \mid v \mid \underline{w}_1)$ and $f(\underline{u} \mid v \mid \underline{w}_2)$ differ and we are done. ◀

The following result makes it easier to compute certain factored outputs.

► **Lemma 47.** *Suppose f is a transduction without data peeking, u, v are data words, $\sigma \in \Sigma$ and $d \in D$. The data values occurring in $f(\underline{u} \mid (\sigma, d) \mid \underline{v})$ are either d or those that are f -suffix influencing in u .*

Proof. From the hypothesis that f is without data peeking, we infer that the data values occurring in $f(\underline{u} \mid (\sigma, d) \mid \underline{v})$ are either d or those that occur in u . Suppose a data value $e \neq d$ occurs in $f(\underline{u} \mid (\sigma, d) \mid \underline{v})$. Let e' be a safe replacement for e in u . We have $f(\underline{u[e/e']} \mid (\sigma, d) \mid \underline{v}) \neq f(\underline{u} \mid (\sigma, d) \mid \underline{v})$, since e cannot occur in $f(\underline{u[e/e']} \mid (\sigma, d) \mid \underline{v})$ but it does occur in $f(\underline{u} \mid (\sigma, d) \mid \underline{v})$. Applying the contrapositive of point 1 in Lemma 44 to the above inequality, we infer that $f(\underline{u[e/e']} \mid (\sigma, d) \cdot v) \neq f(\underline{u} \mid (\sigma, d) \cdot v)$. According to Definition 4, this certifies that e is f -suffix influencing in u . ◀

The following result uses the binary relation \equiv_f from Definition 7 and equalizing schemes from Definition 9.

► **Lemma 48.** *Suppose f is a transduction that is invariant under permutations, E is an equalizing scheme for f and u, u', v, w are data words. If $u \equiv_f u'$, then $f(\underline{E(u)(u)} \mid v \mid \underline{w}) = f_z(\underline{E(u')(u')} \mid v \mid \underline{w})$, where $z = |u| - |u'|$.*

Proof. Since $E(u)(u) \simeq u$, we have $E(u)(u) \equiv_f u$. So we infer that $E(u)(u) \equiv_f u \equiv_f u' \equiv_f E(u')(u')$. Since \equiv_f is transitive, $E(u)(u) \equiv_f E(u')(u')$. So we infer from Definition 7 that there exists a permutation π such that $\pi(\mathbf{aifl}_f(E(u')(u')))) = \mathbf{aifl}_f(E(u)(u))$ and $f(\underline{E(u)(u)} \mid v \cdot w) = f_z(\pi(\underline{E(u')(u')}) \mid v \cdot w)$. Since $u \equiv_f u'$, we infer from Definition 7 and Definition 9 that $\mathbf{aifl}_f(E(u')(u')) = \mathbf{aifl}_f(E(u)(u))$, so π (and hence π^{-1}) is identity on those data values that are f -influencing in $E(u')(u')$. Hence we infer from Lemma 42 that $f_z(\pi(\underline{E(u')(u')}) \mid v \cdot w) = f_z(\pi^{-1} \odot \pi(\underline{E(u')(u')}) \mid v \cdot w) = f_z(\underline{E(u')(u')} \mid v \cdot w)$. Hence, $f(\underline{E(u)(u)} \mid v \cdot w) = f_z(\underline{E(u')(u')} \mid v \cdot w)$. We infer from point 1 of Lemma 44 that $f(\underline{E(u)(u)} \mid v \mid \underline{w}) = f_z(\underline{E(u')(u')} \mid v \mid \underline{w})$. ◀

Suppose a SSRT is at a configuration and reads a data word running a sequence of transitions. If a permutation is applied to the configuration and the data word, then the new data word is read by the SSRT starting from the new configuration running the same sequence of transitions. This is formalized in the following result.

► **Lemma 49.** *Suppose S is a SSRT, the set of registers R is partitioned into two parts R_1, R_2 and $(q, \text{val}_1, n_1), (q, \text{val}_2, n_2)$ are configurations satisfying the following properties:*

- val_1 and val_2 coincide on R_1 ,
- for every $r_1, r_2 \in R$, $\text{val}_1(r_1) = \text{val}_1(r_2)$ iff $\text{val}_2(r_1) = \text{val}_2(r_2)$ and
- $\{\text{val}_1(r) \mid r \in R_1\} \cap \{\text{val}_1(r) \mid r \in R_2\} = \emptyset = \{\text{val}_2(r) \mid r \in R_1\} \cap \{\text{val}_2(r) \mid r \in R_2\}$.

There exists a permutation π that is identity on $\{\text{val}_1(r) \mid r \in R_1\}$ such that for any data word v , the sequence of transitions executed when reading v from (q, val_1) is same as the sequence executed when reading $\pi(v)$ from (q, val_2) .

Proof. Let π be a permutation that is identity on $\{\text{val}_1(r) \mid r \in R_1\}$ such that for every $r_2 \in R_2$, $\pi(\text{val}_1(r_2)) = \text{val}_2(r_2)$. For every register r and every position i of v , $\text{val}_1(r) = \mathbf{data}(v, i)$ iff $\text{val}_2(r) = \mathbf{data}(\pi(v), i)$. The result follows by a routine induction on $|v|$. ◀

The next result says that if two strings belong to the same equivalence class of \equiv_f , then they can be equalized by an equalizing scheme after which both will be transformed similarly by any suffix. It uses the binary relation \equiv_S and the concept of arrangements of elements of a set from Section 5.

► **Lemma 50.** *Suppose S is a SSRT implementing a transduction f , $u_1 \equiv_S u_2$, S reaches the configuration $(q_1, \text{val}_1, |u_1|)$ after reading $E(u_1)(u_1)$ and reaches $(q_2, \text{val}_2, |u_2|)$ after reading $E(u_2)(u_2)$. For any data word v and any i , if the i^{th} left block of $f(\underline{E(u_1)(u_1)} \mid \underline{v})$ is $\text{val}_1(\chi)$ where χ is some arrangement of some subset $X' \subseteq X$, then the i^{th} left block of $f(\underline{E(u_2)(u_2)} \mid \underline{v})$ is $\text{val}_2(\chi)$.*

Proof. Since $u_1 \equiv_S u_2$, $E(u_1)(u_1) \equiv_S E(u_2)(u_2)$, so $q_1 = q_2$, say $q_1 = q_2 = q$. For any i , the i^{th} f -influencing value is δ_i in both $E(u_1)(u_1)$ and $E(u_2)(u_2)$. From condition 3 of Definition 36, we infer that val_1 and val_2 coincide on all the registers that store f -influencing values. Suppose for the sake of contradiction that for some data word v and some i , the i^{th} left block of $f(E(u_1)(u_1) \mid v)$ is $val_1(\chi)$ and the i^{th} left block of $f(E(u_2)(u_2) \mid v)$ is $val_2(\chi') \neq val_2(\chi)$. This means that while reading v from (q, val_2) , the sequence of transitions is different from the sequence when reading v from (q, val_1) . This difference is due to the difference between val_1 and val_2 in registers that don't store f -influencing values. Hence, we infer from Lemma 49 that there exists a permutation π that is identity on f -influencing values such that the sequence of transitions executed when reading v from (q, val_1) is the same sequence executed when reading $\pi(v)$ from (q, val_2) . Hence, the i^{th} left block of $f(E(u_2)(u_2) \mid \pi(v))$ is $val_2(\chi)$, which is different from the i^{th} left block of $f(E(u_2)(u_2) \mid v)$, which is $val_2(\chi')$. Since f is invariant under permutations and without data peeking (from Proposition 34), this contradicts Lemma 43. \blacktriangleleft

B Proofs of Results in Section 3

Proof of Lemma 8. We have $u \equiv_f u$ for all u , since the identity permutation satisfies all the conditions of Definition 7. Hence, \equiv_f is reflexive.

Suppose $u_1 \equiv_f u_2$ and there exists a permutation π satisfying all the conditions of Definition 7. We have $\mathbf{aifl}_f(\pi(u_2)) = \mathbf{aifl}_f(u_1)$ and applying the permutation π^{-1} on both sides gives us $\pi^{-1}(\mathbf{aifl}_f(\pi(u_2))) = \pi^{-1}(\mathbf{aifl}_f(u_1))$. Since f is invariant under permutations, we infer from Lemma 40 that $\mathbf{aifl}_f(u_2) = \mathbf{aifl}_f(\pi^{-1}(u_1))$. For any v , we have $f_z(\pi(u_2) \mid \pi(v)) = f(u_1 \mid \pi(v))$, where $z = |u_1| - |u_2|$. Applying π^{-1} on both sides and using Lemma 39, we get $f_z(\underline{u_2} \mid v) = f(\pi^{-1}(u_1) \mid v)$ for any v . Hence, $\lambda v.f_z(\underline{u_2} \mid v) = \lambda v.f_{-z}(\pi^{-1}(u_1) \mid v)$. For all data words u, v_1, v_2 , we have $f(u_1 \cdot \pi(u) \mid \pi(v_1)) = f(u_1 \cdot \pi(u) \mid \pi(v_2))$ iff $f(\pi(u_2) \cdot \pi(u) \mid \pi(v_1)) = f(\pi(u_2) \cdot \pi(u) \mid \pi(v_2))$. Applying π^{-1} on both sides of both the equalities and using Lemma 39, we get $f(\pi^{-1}(u_1) \cdot u \mid v_1) = f(\pi^{-1}(u_1) \cdot u \mid v_2)$ iff $f(u_2 \cdot u \mid v_1) = f(u_2 \cdot u \mid v_2)$. Hence, π^{-1} satisfies all the conditions of Definition 7, so $u_2 \equiv_f u_1$, so \equiv_f is symmetric.

Suppose $u_1 \equiv_f u_2$ and there exists a permutation π satisfying all the conditions of Definition 7. Suppose $u_2 \equiv_f u_3$ and there exists a permutation π' satisfying all the conditions of Definition 7. Let $\pi \odot \pi'$ be the composition of π and π' ($\pi \odot \pi'(u) = \pi(\pi'(u))$ for all u). It is routine verify the following equalities: $\mathbf{ifl}_f(\pi \odot \pi'(u_3)) = \mathbf{ifl}_f(u_1)$, $\lambda v.f_{z+z'}(\pi \odot \pi'(u_3) \mid v) = f(u_1 \mid v)$ where $z = |u_1| - |u_2|$ and $z' = |u_2| - |u_3|$ and for all data words u, v_1, v_2 , $f(u_1 \cdot u \mid v_1) = f(u_1 \cdot u \mid v_2)$ iff $f(\pi \odot \pi'(u_3) \cdot u \mid v_1) = f(\pi \odot \pi'(u_3) \cdot u \mid v_2)$. Hence \equiv_f is transitive. \blacktriangleleft

Proof of Lemma 11. Let $\delta_1 \delta_2 \dots$ be a sequence of data values such that for every data word u and every i , the i^{th} f -influencing data value of $E_1(u)(u)$ is δ_i . Let $\eta_1 \eta_2 \dots$ be a sequence of data values such that for every data word u and every i , the i^{th} f -influencing data value of $E_2(u)(u)$ is η_i . Let π be a permutation such that $\pi(\delta_1 \delta_2 \dots) = \eta_1 \eta_2 \dots$. Let $\{v_1, \dots, v_m\}$ be a set of data words such that no two of them are in the same equivalence class of $\equiv_f^{E_1}$. We will show that no two data words in $\{\pi(v_1), \dots, \pi(v_m)\}$ are in the same equivalence class of $\equiv_f^{E_2}$.

For all $i, j \in \{1, \dots, m\}$ with $i \neq j$, let u_{ij} be a data word such that $f(E_1(u_{ij})(u_{ij}) \mid v_i) \neq f(E_1(u_{ij})(u_{ij}) \mid v_j)$. Applying the permutation $E_2(u_{ij}) \cdot E_1^{-1}(u_{ij})$ to both sides and using Lemma 39, we infer that $f(E_2(u_{ij})(u_{ij}) \mid E_2(u_{ij}) \cdot E_1^{-1}(u_{ij})(v_i)) \neq f(E_2(u_{ij})(u_{ij}) \mid E_2(u_{ij}) \cdot E_1^{-1}(u_{ij})(v_j))$. Suppose $\mathbf{ifl}_f(E_2(u_{ij})(u_{ij})) = \eta_1 \dots \eta_r$. We will prove that there

exist permutations π_i, π_j such that they are identity on $\eta_1 \cdots \eta_r$, $\pi_i \odot E_2(u_{ij}) \odot E_1^{-1}(u_{ij})(v_i) = \pi(v_i)$ and $\pi_j \odot E_2(u_{ij}) \odot E_1^{-1}(u_{ij})(v_j) = \pi(v_j)$. Then, using Lemma 43, we get

$$\begin{aligned} f(E_2(u_{ij})(u_{ij}) \mid \pi(v_i)) &= f(E_2(u_{ij})(u_{ij}) \mid \pi_i \cdot E_2(u_{ij}) \cdot E_1^{-1}(u_{ij})(v_i)) \\ &= f(E_2(u_{ij})(u_{ij}) \mid E_2(u_{ij}) \cdot E_1^{-1}(u_{ij})(v_i)) \\ &\neq f(E_2(u_{ij})(u_{ij}) \mid E_2(u_{ij}) \cdot E_1^{-1}(u_{ij})(v_j)) \\ &= f(E_2(u_{ij})(u_{ij}) \mid \pi_j \cdot E_2(u_{ij}) \cdot E_1^{-1}(u_{ij})(v_j)) \\ &= f(E_2(u_{ij})(u_{ij}) \mid \pi(v_j)) . \end{aligned}$$

Hence, no two data words in $\{\pi(v_1), \dots, \pi(v_m)\}$ are in the same equivalence class of $\equiv_f^{E_2}$.

Now we will prove that there exists a permutation π_i such that it is identity on $\eta_1 \cdots \eta_r$ and $\pi_i \cdot E_2(u_{ij}) \cdot E_1^{-1}(u_{ij})(v_i) = \pi(v_i)$. Let $\text{ifl}_f(u_{ij}) = d_1 \cdots d_r$. For all $i \in \{1, \dots, r\}$, $\pi : \delta_i \mapsto \eta_i$ and $E_1^{-1}(u_{ij}) : \delta_i \mapsto d_i$, $E_2(u_{ij})(u_{ij}) : d_i \mapsto \eta_i$. Define π_i such that $\pi_i : \eta_i \mapsto \eta_i$. For $\delta \notin \{\delta_1, \dots, \delta_r\}$, suppose $\pi : \delta \mapsto \eta$, $E_1^{-1}(u_{ij}) : \delta \mapsto d$ and $E_2(u_{ij}) : d \mapsto \eta'$. Define π_i such that $\pi_i : \eta' \mapsto \eta$. Now π_i is identity on $\eta_1 \cdots \eta_r$ and $\pi_i \cdot E_2(u_{ij}) \cdot E_1^{-1}(u_{ij})(v_i) = \pi(v_i)$. The existence of π_j can be proved similarly. \blacktriangleleft

C Proofs of Results in Section 4

Proof of Lemma 13. Suppose d is f -suffix influencing in $u \cdot (\sigma, e)$. There exists a data value d' that is a safe replacement for d in $u \cdot (\sigma, e)$ and a data word v such that the next inequality is true.

$$\begin{aligned} f((u \cdot (\sigma, e))[d/d'] \mid v) &\neq f(u \cdot (\sigma, e) \mid v) \\ f(u[d/d'] \cdot (\sigma, e) \mid v) &\neq f(u \cdot (\sigma, e) \mid v) \quad [d \neq e] \\ f(u[d/d'] \mid (\sigma, e) \cdot v) &\neq f(u \mid (\sigma, e) \cdot v) \quad [\text{contrapositive of Lemma 44, point 2}] \end{aligned}$$

The last inequality above shows that d is f -suffix influencing in u .

Suppose d is f -prefix influencing in $u \cdot (\sigma, e)$. Then there exist data words u', v and a data value d' such that d doesn't occur in u' , d' is a safe replacement for d in $u \cdot (\sigma, e) \cdot u' \cdot v$ and $f(u \cdot (\sigma, e) \cdot u' \mid v[d/d']) \neq f(u \cdot (\sigma, e) \cdot u' \mid v)$. Since d doesn't occur in u' and $d \neq e$, d doesn't occur in $(\sigma, e) \cdot u'$. We observe that $f(u \cdot ((\sigma, e) \cdot u') \mid v[d/d']) \neq f(u \cdot ((\sigma, e) \cdot u') \mid v)$ to conclude that d is f -prefix influencing in u . \blacktriangleleft

Proof of Lemma 16. By induction on $|u|$. The base case with $|u| = 0$ is trivial. As induction hypothesis, suppose that after reading a data word u , the SSRT reaches the configuration $(([u]_f, ptr), val, |u|)$ such that $val(ptr(i))$ is the i^{th} f -influencing value in u for all $i \in [1, m]$, where $m = |\text{ifl}_f(u)|$. Suppose the SSRT reads $(\sigma, d) \in \Sigma \times D$ next. We give the proof for the case where d is not f -influencing in u and it is f -influencing in $u \cdot (\sigma, d)$. The other cases are similar. Let m' be the number of f -influencing values in $E(u)(u) \cdot (\sigma, \delta_0)$. We infer from Lemma 14 that δ_0 is f -influencing in $E(u)(u) \cdot (\sigma, \delta_0)$. We prove that the transition from $([u]_f, ptr)$ corresponding to $i = 0$ in Construction 15 can be executed. We infer from Lemma 14 that $u \cdot (\sigma, d) \equiv_f E(u)(u) \cdot (\sigma, \delta_0)$ so $[u \cdot (\sigma, d)]_f = [E(u)(u) \cdot (\sigma, \delta_0)]_f$, the next state of the SSRT. The condition $\phi = \bigwedge_{j=1}^{j=m} ptr(j) \neq$ is satisfied since d is not f -influencing in u and for all $j \in [1, m]$, $val(ptr(j))$ is the j^{th} f -influencing value in u , which is not equal to d . We infer from Lemma 14 that $u \cdot (\sigma, d)$ has m' f -influencing values. For every $j \in [1, m]$, δ_j is f -influencing in $E(u)(u) \cdot (\sigma, \delta_0)$ iff the j^{th} f -influencing value in u (which is assigned to $ptr(j)$ by val) is f -influencing in $u \cdot (\sigma, d)$. Since δ_0 is the 1^{st} f -influencing

value in $E(u)(u) \cdot (\sigma, \delta_0)$, $ptr'(1) = r_{reuse}$ as given in Construction 15. Since r_{reuse} is the first register in the set $R \setminus \{ptr(l) \mid 1 \leq l \leq m, \delta_l \text{ is } f\text{-influencing in } E(u)(u) \cdot (\sigma, \delta_0)\}$, r_{reuse} is the first register that is not holding a data value that is f -influencing in u and in $u \cdot (\sigma, d)$. Since $R' = \{r_{reuse}\}$, the transition of the SSRT changes the valuation to val' such that $val'(r_{reuse}) = d$. So $val'(ptr'(1)) = val'(r_{reuse}) = d$, the first f -influencing value in $u \cdot (\sigma, d)$. Suppose $j \in [2, m']$ and the j^{th} f -influencing value in $E(u)(u) \cdot (\sigma, \delta_0)$ is δ_k , the k^{th} f -influencing value in $E(u)(u)$ (this will be true for some k , by Lemma 13). Since $R = \{r_{reuse}\}$, val and val' coincide on all registers except r_{reuse} . Since r_{reuse} is the first register in the set $R \setminus \{ptr(l) \mid 1 \leq l \leq m, \delta_l \text{ is } f\text{-influencing in } E(u)(u) \cdot (\sigma, \delta_0)\}$, $r_{reuse} \neq ptr(k)$ and val and val' coincide on $ptr(k)$. Hence, $val'(ptr(k)) = val(ptr(k))$. Since the j^{th} f -influencing value in $E(u)(u) \cdot (\sigma, \delta_0)$ is δ_k , the k^{th} f -influencing value in $E(u)(u)$, we infer from Lemma 14 that the j^{th} f -influencing value in $u \cdot (\sigma, d)$ is the k^{th} f -influencing value in u . Hence, $val'(ptr'(j)) = val'(ptr(k)) = val(ptr(k))$, which is the k^{th} f -influencing value in u and the j^{th} f -influencing value in $u \cdot (\sigma, d)$. The first equality above follows since $ptr'(j) = ptr(k)$ as given in Construction 15. \blacktriangleleft

Proof of Lemma 20. Since $E(u \cdot (\sigma, d))^{-1}(v)$ and $E(u)^{-1}(\pi(v))$ are obtained from applying different permutations to v , they are isomorphic. We will prove that for all j and $i \geq 2$, if the j^{th} position of $E(u \cdot (\sigma, d))^{-1}(v)$ contains the i^{th} f -influencing value of $u \cdot (\sigma, d)$, then the same is contained in the j^{th} position of $E(u)^{-1}(\pi(v))$.

1. The j^{th} position of $E(u \cdot (\sigma, d))^{-1}(v)$ contains the i^{th} f -influencing value of $u \cdot (\sigma, d)$.
2. Hence, the j^{th} position of v contains δ_i , by definition of equalizing schemes (Definition 9).
3. For $i \geq 2$, the i^{th} f -influencing value of $u \cdot (\sigma, d)$ is among $\{d_m, \dots, d_1\}$, the f -influencing values in u , by Lemma 13.
4. Say d_k is the i^{th} f -influencing value of $u \cdot (\sigma, d)$. Then δ_k is the i^{th} f -influencing value of $E(u')(u') \cdot (\sigma, \eta)$, by Lemma 14.
5. The permutation π maps δ_i to δ_k , by Definition 19.
6. The permutation $E(u)^{-1}$ maps δ_k to d_k , by definition of equalizing schemes (Definition 9).
7. The j^{th} position of $E(u)^{-1}(\pi(v))$ contains d_k , by points (2), (5) and (6) above.
8. By point (4) above, d_k is the i^{th} f -influencing value of $u \cdot (\sigma, d)$, so the j^{th} position of $E(u)^{-1}(\pi(v))$ contains the i^{th} f -influencing value of $u \cdot (\sigma, d)$.

Suppose $(d, \eta) \in \{(d_i, \delta_i) \mid i \in \{1, \dots, m\}\}$ or $(d, \eta) = (d_0, \delta_0)$ and the first f -influencing value in $u \cdot (\sigma, d)$ is among $\{d_m, \dots, d_1\}$. Then we can put $i \geq 1$ in the above reasoning to infer that for all j and $i \geq 1$, if the j^{th} position of $E(u \cdot (\sigma, d))^{-1}(v)$ contains the i^{th} f -influencing value of $u \cdot (\sigma, d)$, then the same is contained in the j^{th} position of $E(u)^{-1}(\pi(v))$. Hence, we get the following equality.

$$\begin{aligned}
f(u \cdot (\sigma, d) \mid E(u \cdot (\sigma, d))^{-1}(v)) &= f(u \cdot (\sigma, d) \mid E(u)^{-1}(\pi(v))) && [\text{Lemma 43}] \\
f(u \mid (\sigma, d) \cdot E(u \cdot (\sigma, d))^{-1}(v)) &= f(u \mid (\sigma, d) \cdot E(u)^{-1}(\pi(v))) && [\text{Lemma 44, point 2}] \\
f(u \mid (\sigma, d) \cdot E(u \cdot (\sigma, d))^{-1}(v)) &= f(u \mid E(u)^{-1}((\sigma, \eta) \cdot \pi(v))) && [E(u)(d) = \eta]
\end{aligned}$$

Suppose $(d, \eta) = (d_0, \delta_0)$ and the first f -influencing value in $u \cdot (\sigma, d)$ is d . Then π maps δ_1 to $\eta = \delta_0$. Let π' be the permutation that interchanges $E(u)^{-1}(\eta)$ and d and doesn't change any other value. For all j and $i \geq 1$, if the j^{th} position of $E(u \cdot (\sigma, d))^{-1}(v)$ contains the i^{th} f -influencing value of $u \cdot (\sigma, d)$, then the same is contained in the j^{th} position of

$\pi' \odot E(u)^{-1}(\pi(v))$. Hence, we get the following equality.

$$\begin{aligned} f(u \cdot (\sigma, d) \mid \underline{E(u \cdot (\sigma, d))^{-1}(v)}) &= f(u \cdot (\sigma, d) \mid \underline{\pi' \odot E(u)^{-1}(\pi(v))}) \quad [\text{Lemma 42}] \\ f(u \mid \underline{(\sigma, d) \cdot E(u \cdot (\sigma, d))^{-1}(v)}) &= f(u \mid \underline{(\sigma, d) \cdot \pi' \odot E(u)^{-1}(\pi(v))}) \quad [\text{Lemma 44, point 2}] \end{aligned}$$

Since $\eta = \delta_0$ does not occur in $\{\delta_m, \dots, \delta_1\}$, $E(u)^{-1}(\eta)$ does not occur in $\{d_m, \dots, d_1\}$, the f -influencing data values in u . Since d also does not occur in $\{d_m, \dots, d_1\}$, π' only interchanges two data values that are not f -influencing in u and doesn't change any other value. So we infer from Lemma 43 that $f(u \mid (\sigma, d) \cdot \pi' \odot E(u)^{-1}(\pi(v))) = f(u \mid \pi'((\sigma, d)) \cdot \pi' \odot \pi' \odot E(u)^{-1}(\pi(v))) = f(u \mid E(u)^{-1}((\sigma, \eta)) \cdot E(u)^{-1}(\pi(v))) = f(u \mid \underline{E(u)^{-1}((\sigma, \eta) \cdot \pi(v))})$. Combining this with the equality above, we get $f(u \mid (\sigma, d) \cdot E(u \cdot (\sigma, d))^{-1}(v)) = f(u \mid \underline{E(u)^{-1}((\sigma, \eta) \cdot \pi(v))})$.

Now we will prove the statements about $f(u \mid (\sigma, d) \mid \underline{E(u \cdot (\sigma, d))^{-1}(v)})$. Let g be a function such that for $i \geq 2$, the i^{th} f -influencing value in $E(u)(u \cdot (\sigma, \eta))$ is $\delta_{g(i)}$.

Case 1: $(d, \eta) \in \{(d_i, \delta_i) \mid i \in [1, m]\}$. Let $\text{ifl}_f(E(u \cdot (\sigma, d))(u \cdot (\sigma, d))) = \delta_r \cdots \delta_1$. We will first prove that $E(u) \odot E(u \cdot (\sigma, d))^{-1}$ coincides with π on $\delta_r, \dots, \delta_1$. For $i \geq 2$, $E(u \cdot (\sigma, d))^{-1}(\delta_i)$ is the i^{th} f -influencing value in $u \cdot (\sigma, d)$ and we infer from Lemma 14 that the i^{th} f -influencing value in $u \cdot (\sigma, d)$ is $d_{g(i)}$, the $g(i)^{\text{th}}$ f -influencing value in u (since the i^{th} f -influencing value in $E(u)(u \cdot (\sigma, \eta))$ is $\delta_{g(i)}$, the $g(i)^{\text{th}}$ f -influencing value in $E(u)(u)$). By Definition 9, $E(u)$ maps $d_{g(i)}$ to $\delta_{g(i)}$. Hence, for $i \geq 2$, $E(u) \odot E(u \cdot (\sigma, d))^{-1}$ maps δ_i to $\delta_{g(i)}$, which is exactly what π does to δ_i . Say the first f -influencing value in $E(u')(u') \cdot (\sigma, \eta)$ is δ_j . We infer from Lemma 14 that the first f -influencing value in $u \cdot (\sigma, d)$ is d_j . Hence, $E(u) \odot E(u \cdot (\sigma, d))^{-1}$ maps δ_1 to δ_j , which is exactly what π does to δ_1 . Hence, $E(u) \odot E(u \cdot (\sigma, d))^{-1}$ coincides with π on $\delta_r, \dots, \delta_1$, the f -influencing values of $E(u \cdot (\sigma, d))(u \cdot (\sigma, d))$.

$$\begin{aligned} E(u)(f(u \mid (\sigma, d) \mid \underline{E(u \cdot (\sigma, d))^{-1}(v)})) &= E(u) \odot E(u \cdot (\sigma, d))^{-1} \odot E(u \cdot (\sigma, d))(f(u \mid (\sigma, d) \mid \underline{E(u \cdot (\sigma, d))^{-1}(v)})) \\ &= E(u) \odot E(u \cdot (\sigma, d))^{-1}(f(\underline{E(u \cdot (\sigma, d))}(u) \mid \underline{E(u \cdot (\sigma, d))(\sigma, d)} \mid \underline{v})) \quad [\text{Lemma 39}] \\ &= f(\underline{E(u)(u)} \mid \underline{E(u)(\sigma, d)} \mid \underline{\pi(v)}) \quad [\text{Lemma 45}] \\ &= f_z(\underline{E(u')(u')} \mid (\sigma, \eta) \mid \underline{\pi(v)}) \quad [\text{Lemma 48}] \end{aligned}$$

In the last inequality above, apart from Lemma 48, we also use the fact that $E(u)(d) = E(u)(d_i) = \delta_i = \eta$. So we get $E(u)(f(u \mid (\sigma, d) \mid \underline{E(u \cdot (\sigma, d))^{-1}(v)})) = f_z(\underline{E(u')(u')} \mid (\sigma, \eta) \mid \underline{\pi(v)})$, concluding the proof for this case.

Case 2: $(d, \eta) = (d_0, \delta_0)$. Let π_1 be any permutation satisfying the following conditions:

- For $i \geq 2$, $\pi_1(\delta_i) = \pi(\delta_i)$,
- if the first f -influencing value in $E(u')(u') \cdot (\sigma, \eta)$ is δ_j for some $j \geq 1$, then $\pi_1(\delta_1) = \pi(\delta_1)$ and
- if the first f -influencing value in $E(u')(u') \cdot (\sigma, \eta)$ is $\eta = \delta_0$, then $\pi_1(\delta_1) = E(u)(d) = E(u)(d_0)$.

As seen in case 1, $E(u) \cdot E(u \cdot (\sigma, d))^{-1}$ coincides with π_1 on $\delta_r, \dots, \delta_2$. If the first f -influencing value in $E(u)(u \cdot (\sigma, \eta))$ is δ_j for some $j \geq 1$, then again as in case 1, $E(u) \cdot E(u \cdot (\sigma, d))^{-1}$ coincides with π_1 on δ_1 . If the first f -influencing value in $E(u)(u \cdot (\sigma, \eta))$ is η , we infer from Lemma 14 that the first f -influencing value in $u \cdot (\sigma, d)$ is d , so $E(u \cdot (\sigma, d))^{-1}$ maps δ_1 to d . In this case, $\pi_1(\delta_1) = E(u)(d)$, so $E(u) \cdot E(u \cdot (\sigma, d))^{-1}$ coincides with π_1 on δ_1 .

So $E(u) \cdot E(u \cdot (\sigma, d))^{-1}$ coincides with π_1 on $\delta_r, \dots, \delta_1$. Hence, similar to case 1, we get $E(u)(f(\underline{u} \mid (\sigma, d) \mid E(u \cdot (\sigma, d))^{-1}(v))) = f_z(E(u')(u') \mid E(u)(\sigma, d) \mid \pi_1(v))$.

Recall that δ_0 is a data value that is not f -influencing in $E(u')(u')$ and does not occur in $\{\delta_m, \dots, \delta_1\}$. Let π' be the permutation that interchanges δ_0 and $E(u)(d)$ and doesn't change any other value. Since d is not f -influencing in u , $E(u)(d)$ does not occur in $\{\delta_m, \dots, \delta_1\}$. Since the f -influencing values of $E(u')(u')$ are $\delta_m, \dots, \delta_1$ and neither δ_0 nor $E(u)(d)$ occur in $\{\delta_m, \dots, \delta_1\}$, we get the following:

$$\begin{aligned}
f(E(u')(u') \mid (\sigma, \delta_0) \cdot \pi' \odot \pi_1(v)) &= f(\pi' \odot E(u')(u') \mid (\sigma, \delta_0) \cdot \pi' \odot \pi_1(v)) && \text{[Lemma 42]} \\
f(E(u')(u') \mid (\sigma, \delta_0) \mid \pi' \odot \pi_1(v)) &= f(\pi' \odot E(u')(u') \mid (\sigma, \delta_0) \mid \pi' \odot \pi_1(v)) && \text{[Lemma 44, point 1]} \\
E(u)(f(\underline{u} \mid (\sigma, d) \mid E(u \cdot (\sigma, d))^{-1}(v))) &= f_z(E(u')(u') \mid E(u)(\sigma, d) \mid \pi_1(v)) \\
\pi' \odot E(u)(f(\underline{u} \mid (\sigma, d) \mid E(u \cdot (\sigma, d))^{-1}(v))) &= \pi'(f_z(E(u')(u') \mid E(u)(\sigma, d) \mid \pi_1(v))) && \text{[apply } \pi' \text{ on both sides]} \\
&= f_z(\pi' \odot E(u')(u') \mid (\sigma, \delta_0) \mid \pi' \odot \pi_1(v)) && \text{[Lemma 39]} \\
&= f_z(E(u')(u') \mid (\sigma, \delta_0) \mid \pi' \odot \pi_1(v)) && \text{[second equality above]}
\end{aligned}$$

For $i \in \{1, \dots, r\}$, π maps δ_i to the i^{th} f -influencing value in $E(u')(u') \cdot (\sigma, \eta)$ by definition. We will prove that $\pi' \odot \pi_1$ does exactly the same on $\delta_1, \dots, \delta_r$. For $i \geq 2$, π maps δ_i to the i^{th} f -influencing value in $E(u')(u') \cdot (\sigma, \eta)$, which is among $\delta_m, \dots, \delta_1$. By definition, π_1 also maps δ_i to the i^{th} f -influencing value in $E(u')(u') \cdot (\sigma, \eta)$, and π' doesn't change this value, since neither $E(u)(d)$ nor δ_0 are among $\delta_m, \dots, \delta_1$. The permutation π maps δ_1 to the first f -influencing value in $E(u')(u') \cdot (\sigma, \eta)$. If this first f -influencing value is δ_j for some $j \geq 1$, then, by definition, π_1 also maps δ_1 to the first f -influencing value in $E(u')(u') \cdot (\sigma, \eta)$, and π' doesn't change this value, since neither $E(u)(d)$ nor δ_0 are among $\delta_m, \dots, \delta_1$. If the first f -influencing value in $E(u')(u') \cdot (\sigma, \eta)$ is $\eta = \delta_0$, then π maps δ_1 to δ_0 . By definition, π_1 maps δ_1 to $E(u)(d)$ and π' maps $E(u)(d)$ to δ_0 . Hence, $\pi' \odot \pi_1$ maps δ_1 to δ_0 . Therefore, for $i \in \{1, \dots, r\}$, both π and $\pi' \odot \pi_1$ map δ_i to the i^{th} f -influencing value in $E(u')(u') \cdot (\sigma, \eta)$. Hence, we can apply Lemma 43 to get the next equality.

$$\begin{aligned}
f(E(u')(u') \cdot (\sigma, \delta_0) \mid \pi' \odot \pi_1(v)) &= f(E(u')(u') \cdot (\sigma, \delta_0) \mid \pi(v)) \\
f(E(u')(u') \mid (\sigma, \delta_0) \mid \pi' \odot \pi_1(v)) &= f(E(u')(u') \mid (\sigma, \delta_0) \mid \pi(v)) && \text{[Lemma 44, point 4]}
\end{aligned}$$

Hence $\pi' \odot E(u)(f(\underline{u} \mid (\sigma, d) \mid E(u \cdot (\sigma, d))^{-1}(v))) = f_z(E(u')(u') \mid (\sigma, \delta_0) \mid \pi(v))$, concluding the proof for this case. \blacktriangleleft

Proof of Lemma 25. Suppose $[v]_f^E$ is an equivalence class and θ_v, θ are as explained in Definition 24. If d is the i^{th} f -influencing value in u for some $i \geq 1$, let $\eta = \delta_i$ and let $\eta = \delta_0$ otherwise. Let u' be an arbitrary data word in $[u]_f$. We have from Lemma 14 that $u \cdot (\sigma, d) \equiv_f E(u')(u') \cdot (\sigma, \eta)$, so $\text{pref}(\theta) = [E(u')(u') \cdot (\sigma, \eta)]_f = [u \cdot (\sigma, d)]_f$ as required. We have from Lemma 20 that $f(\underline{u} \mid (\sigma, d) \mid E(u \cdot (\sigma, d))^{-1}(v))$ is equal to either $E(u)^{-1}(f_z(E(u')(u') \mid (\sigma, \eta) \mid \pi(v)))$ or $E(u)^{-1} \odot \pi'(f_z(E(u')(u') \mid (\sigma, \eta) \mid \pi(v)))$. Hence, $f(\underline{u} \mid (\sigma, d) \mid E(u \cdot (\sigma, d))^{-1}(v))$ and $f_z(E(u')(u') \mid (\sigma, \eta) \mid \pi(v))$ are isomorphic. Hence, the i^{th} left block of $f(u \cdot (\sigma, d) \mid E(u \cdot (\sigma, d))^{-1}(v))$ is the concretization of z , the i^{th} non-right block of $f(E(u')(u') \mid (\sigma, \eta) \mid \pi(v))$, as defined in Definition 24. We will prove that $\text{val}'(\text{ur}(\theta, \text{bl}(\theta, i)))$ is the concretization of z , which is sufficient to complete the proof.

Indeed, $\text{val}'(\text{ur}(\theta, \text{bl}(\theta, i))) = \text{val}'(\text{ur}(\theta, z'))$, where z' is obtained from z by replacing j^{th} left block by P_j and k^{th} middle block by $\langle \theta, k \rangle$. Since we set $\text{val}'(\langle \theta, k \rangle)$ to be the k^{th} middle block of $f(\underline{u} \mid (\sigma, d) \mid E(u \cdot (\sigma, d))^{-1}(v))$, $\text{val}'(\text{ur}(\theta, \text{bl}(\theta, i)))$ correctly concretizes the middle blocks. Since $\text{ur}(\theta, P_j) = \text{ur}(\theta_v, \text{bl}(\theta_v, j))$ and θ_v is a node in the original tree T ,

we infer that $\text{val}(\text{ur}(\theta_v, \text{bl}(\theta_v, j)))$ is the j^{th} left block of $f(u \mid \overline{E(u)^{-1}((\sigma, \eta) \cdot \pi(v))})$. Since val and val' differ only in the variables $\langle \theta, k \rangle$ where θ is newly introduced, we infer that $\text{val}'(\text{ur}(\theta_v, \text{bl}(\theta_v, j))) = \text{val}(\text{ur}(\theta_v, \text{bl}(\theta_v, j)))$ is the j^{th} left block of $f(u \mid \overline{E(u)^{-1}((\sigma, \eta) \cdot \pi(v))})$. From Lemma 20, we infer that the j^{th} left block of $f(u \mid \overline{E(u)^{-1}((\sigma, \eta) \cdot \pi(v))})$ is equal to the j^{th} left block of $f(u \mid \overline{(\sigma, d) \cdot E(u \cdot (\sigma, d))^{-1}(v)})$. Hence, $\text{val}'(\text{ur}(\theta, \text{bl}(\theta, j)))$ correctly concretizes the left blocks. \blacktriangleleft

Proof of Lemma 27. Suppose T' is obtained from T by removing a node θ and making the only child of θ a child of θ 's parent. If the only child of θ is $\theta \cdot [v]_f^E$, we will prove that for all $i \in [1, B]$, $\text{ur}(\theta \leftarrow \cdot [v]_f^E, \text{bl}(\theta \leftarrow \cdot [v]_f^E, i)) = \text{ur}(\theta \cdot [v]_f^E, \text{bl}(\theta \cdot [v]_f^E, i))$. This will imply that the unrolling of any block description in any leaf remains unchanged due to the shortening, so the lemma will be proved. First we will prove that $\text{ur}(\theta \leftarrow \cdot [v]_f^E, \text{bl}(\theta, j)) = \text{ur}(\theta, \text{bl}(\theta, j))$. Indeed, both are obtained from $\text{bl}(\theta, j)$ by replacing every occurrence of P_k by $\text{ur}(\theta \leftarrow \cdot, \text{bl}(\theta \leftarrow \cdot, k))$.

We get $\text{ur}(\theta \cdot [v]_f^E, \text{bl}(\theta \cdot [v]_f^E, i))$ from $\text{bl}(\theta \cdot [v]_f^E, i)$ by replacing every occurrence of P_j by $\text{ur}(\theta, \text{bl}(\theta, j))$. We will prove that we also get $\text{ur}(\theta \leftarrow \cdot [v]_f^E, \text{bl}(\theta \leftarrow \cdot [v]_f^E, i))$ from $\text{bl}(\theta \cdot [v]_f^E, i)$ by replacing every occurrence of P_j by $\text{ur}(\theta, \text{bl}(\theta, j))$, which is sufficient to prove the lemma.

Recall that $\text{bl}(\theta \leftarrow \cdot [v]_f^E, i)$ is obtained from $\text{bl}(\theta \cdot [v]_f^E, i)$ by replacing every occurrence of P_j by $\text{bl}(\theta, j)$, as given in Definition 26. Hence, we get $\text{ur}(\theta \leftarrow \cdot [v]_f^E, \text{bl}(\theta \leftarrow \cdot [v]_f^E, i))$ from $\text{bl}(\theta \cdot [v]_f^E, i)$ by first replacing every occurrence of P_j by $\text{bl}(\theta, j)$, which is then replaced by $\text{ur}(\theta \leftarrow \cdot [v]_f^E, \text{bl}(\theta, j)) = \text{ur}(\theta, \text{bl}(\theta, j))$. Hence, for all $i \in [1, B]$, $\text{ur}(\theta \leftarrow \cdot [v]_f^E, \text{bl}(\theta \leftarrow \cdot [v]_f^E, i)) = \text{ur}(\theta \cdot [v]_f^E, \text{bl}(\theta \cdot [v]_f^E, i))$. \blacktriangleleft

Proof of Lemma 32. Since S is an extension of the SSRT constructed in Construction 15, the claim about the pointer function ptr comes from Lemma 16. For the We will prove that (T, val) is complete for u by induction on $|u|$. For the base case, $|u| = 0$ and we infer that $(([e]_f, \text{ptr}_\perp, T_\perp), \text{val}_\epsilon)$ is complete for $u = \epsilon$ by definition. We inductively assume that after reading u , S reaches the configuration $(([u]_f, \text{ptr}, T), \text{val}, |u|)$ such that $\text{val}(\text{ptr}(i))$ is the i^{th} f -influencing value in u and (T, val) is complete for u . Suppose the next symbol read by the SSRT is (σ, d) and $m = |\text{ifl}_f(u)|$.

If d is the i^{th} f -influencing value in u for some $i \geq 1$, let $\eta = \delta_i$ and let $\eta = \delta_0$ otherwise. Let π be a permutation tracking influencing values on $E(u')(u') \cdot (\sigma, \eta)$ as given in Definition 19. Suppose T_1 is the (σ, η) extension of T , T_2 is obtained from T_1 by shortening it as much as possible and T' is the trimming of T_2 . Let ud_1 be the function as defined in Construction 31. If S had the transition $(([u]_f, \text{ptr}, T), \sigma, \phi, ([E(u')(u') \cdot (\sigma, \eta)]_f, \text{ptr}', T_1), R', ud_1)$, S would read (σ, d) and reach the configuration $(([E(u')(u') \cdot (\sigma, \eta)]_f, \text{ptr}', T_1), \text{val}_1, |u| + 1)$. We will prove that (T_1, val_1) is complete for $u \cdot (\sigma, d)$. This can be inferred from Lemma 25 if val_1 is the (σ, d) extension of (T, val) . This can be inferred if val_1 is obtained from val by setting $\text{val}_1(\langle \theta, k \rangle)$ to the k^{th} middle block of $f(u \mid (\sigma, d) \mid \overline{E(u \cdot (\sigma, d))^{-1}(v)})$ for every leaf $\theta = \theta_v \cdot [v]_f^E$ that is newly added while extending T to T_1 . This can be inferred from Lemma 20 if $\text{val}_1(\langle \theta, k \rangle)$ is set to z_1 , the k^{th} middle block of $E(u)^{-1}(f_z(\overline{E(u')(u') \mid (\sigma, \eta) \mid \pi(v)}))$ if $\eta = \delta_i$ for some $i \in [1, m]$ and $\text{val}_1(\langle \theta, k \rangle)$ is set to z_2 , the k^{th} middle block of $E(u)^{-1} \odot \pi'(f_z(\overline{E(u')(u') \mid (\sigma, \eta) \mid \pi(v)}))$ if $\eta = \delta_0$, where $z = |u| - |u'|$ and π' is the permutation that interchanges δ_0 and $E(u)(d)$ and doesn't change any other value. From the semantics of SSRTs, we infer that the third component in every triple of $\text{val}_1(\langle \theta, k \rangle)$ is $|u| + 1$, as required. Hence, it remains to prove that $\downarrow_2(\text{val}_1(\langle \theta, k \rangle)) = \downarrow_2(z_1)$ if $\eta = \delta_i$ for some $i \in [1, m]$ and $\downarrow_2(\text{val}_1(\langle \theta, k \rangle)) = \downarrow_2(z_2)$ if $\eta = \delta_0$.

From Lemma 47, we infer that all data values in $f_z(\overline{E(u')(u') \mid (\sigma, \eta) \mid \pi(v)})$ are among $\{\delta_0, \dots, \delta_m\}$. Hence, we get z_1 and z_2 from the k^{th} middle block of $f_z(\overline{E(u')(u') \mid (\sigma, \eta) \mid \pi(v)})$ by replacing every occurrence of δ_j for $j \in [1, m]$ by $E(u)^{-1}(\delta_j)$ (which is the j^{th} f -influencing

value in u) and replacing every occurrence of δ_0 by $E(u)^{-1} \odot \pi'(\delta_0)$ (which is d). This exactly what the update function ud_1 does to $\langle \theta, k \rangle$: it is set to the k^{th} middle block of $f_z(E(u')(u') \mid (\sigma, \eta) \mid \pi(v))$ and every occurrence of δ_j is replaced by $ptr(j)$ (the transition of S then replaces this with $val(ptr(j))$, the j^{th} f -influencing value in u) and every occurrence of δ_0 is replaced by $curr$ (the transition of S then replaces this with d , the current data value being read). Hence, (T_1, val_1) is complete for $u \cdot (\sigma, d)$.

Since T_2 is obtained from T_1 by shortening it as much as possible, we infer from Lemma 27 that (T_2, val_1) is complete for $u \cdot (\sigma, d)$. The actual transition in S is $(([u]_f, ptr, T), \sigma, \phi, ([E(u')(u') \cdot (\sigma, \eta)]_f, ptr', T'), R', ud)$. After reading (σ, d) , S goes to the configuration $(([u \cdot (\sigma, \eta)]_f, ptr', T'), val', |u| + 1)$ where val' is the trimming of val_1 (due to the way ud is defined from ud_1). Since T' is the trimming of T_2 , we conclude from Proposition 29 that (T', val') is complete for $u \cdot (\sigma, d)$. ◀

D Proofs of Results in Section 5

Proof of Lemma 35. Suppose a data value d is not stored in any of the registers after reading u . We will prove that d is neither f -suffix influencing nor f -prefix influencing in u . To prove that d is not f -suffix influencing in u , we will show that for any data word v and any safe replacement d' for d in u , $f(u[d/d'] \mid v) = f(u \mid v)$. Indeed, let π be the permutation that interchanges d and d' and that doesn't change any other value. We have $u[d/d'] = \pi(u)$. Suppose S reaches the configuration (q, val) after reading u . We infer from Lemma 33 that S reaches the configuration $(q, \pi(val))$ after reading $\pi(u)$. Since d is not stored in any of the registers under the valuation val , $\pi(val)$ coincides with val on all registers. Hence, if S executes a sequence of transitions reading a data word v from the configuration (q, val) , the same sequence of transitions are executed reading v from $(q, \pi(val))$. Since $f(u[d/d'] \mid v)$ and $f(u \mid v)$ depends only on the sequence of transitions that are executed while reading v , we infer that $f(u[d/d'] \mid v) = f(u \mid v)$.

Next we will prove that if a data value d is not stored in any of the registers after reading u , then d is not f -prefix influencing in u . Let u', v be data words and d' be a data value such that d doesn't occur in u' . Since d is not stored in any of the registers after reading u and d doesn't occur in u' , d is not stored in any of the registers after reading $u \cdot u'$. Suppose d' is a safe replacement for d in $u \cdot u' \cdot v$. Then d' doesn't occur in $u \cdot u'$ so neither d' nor d is stored in any of the registers after reading $u \cdot u'$. Since d' doesn't occur in v , $v \simeq v[d/d']$. Hence the SSRT executes the same sequence of transitions for reading $u \cdot u' \cdot v$ and for $u \cdot u' \cdot v[d/d']$. Hence, the only difference between $f(u \cdot u' \cdot v)$ and $f(u \cdot u' \cdot v[d/d'])$ is that at some positions whose origin is not in $u \cdot u'$, the first one may contain d and the second one may contain d' . Since such positions are abstracted out, $f(u \cdot u' \mid v[d/d']) = f(u \cdot u' \mid v)$. Hence, d is not f -prefix influencing in u . ◀

Proof of Lemma 37. We will prove that \equiv_S refines \equiv_f . Suppose u_1, u_2 are data words such that $u_1 \equiv_S u_2$ and S reaches the configurations (q, val_1) , (q, val_2) after reading u_1, u_2 respectively. Let π be a permutation such that for every register r , $\pi(val_2(r)) = val_1(r)$. We can verify by a routine induction on $|u_2|$ that after reading $\pi(u_2)$, S reaches the configuration $(q, \pi(val_2))$. We infer from Lemma 35 that all f -influencing values of u_1 are stored in registers in the configuration (q, val_1) and all f -influencing values of $\pi(u_2)$ are stored in registers in the configuration $(q, \pi(val_2))$. The valuations $\pi(val_2)$ and val_1 coincide on all the registers. Hence, we can infer from condition 3 of Definition 36 that $\text{aifl}_f(\pi(u_2)) = \text{aifl}_f(u_1)$.

Since $\pi(val_2)$ and val_1 coincide on all the registers, for any data word v , the sequence of transitions executed when reading v from the configuration (q, val_1) and from $(q, \pi(val_2))$ are the same. Hence, $f_z(\pi(u_2) \mid v) = f_z(u_1 \mid v)$, where $z = |u_1| - |u_2|$.

Let u, v_1, v_2 be data words. To finish the proof, we have to show that $f(u_1 \cdot u \mid \underline{v_1}) = f(u_1 \cdot u \mid \underline{v_2})$ iff $f(\pi(u_2) \cdot u \mid \underline{v_1}) = f(\pi(u_2) \cdot u \mid \underline{v_2})$. Any left factor of $f(u_1 \mid u \cdot \underline{v_1})$ is of the form $val_1(\chi)$, where χ_1 is some arrangement of some subset $X_1 \subseteq X$. Since val_1 and $\pi(val_2)$ coincide on all the registers and $val_1(x) = \epsilon$ iff $\pi(val_2)(x) = \epsilon$ for all data word variables $x \in X$ (by condition 4 of Definition 36), it can be routinely verified that $f(u_1 \mid u \cdot \underline{v_1})$ and $f(\pi(u_2) \mid u \cdot \underline{v_1})$ have the same number of left blocks and right blocks. If the i^{th} left block of $f(u_1 \mid u \cdot \underline{v_1})$ is $val_1(\chi)$, then the i^{th} left block of $f(\pi(u_2) \mid u \cdot \underline{v_1})$ is $\pi(val_2)(\chi)$. We will assume that $f(u_1 \cdot u \mid \underline{v_1}) \neq f(u_1 \cdot u \mid \underline{v_2})$ and show that $f(\pi(u_2) \cdot u \mid \underline{v_1}) \neq f(\pi(u_2) \cdot u \mid \underline{v_2})$. The proof of the converse direction is symmetric. It is sufficient to prove that either $f(\pi(u_2) \mid u \cdot \underline{v_1}) \neq f(\pi(u_2) \mid u \cdot \underline{v_2})$ or $f(\pi(u_2) \mid u \mid \underline{v_1}) \neq f(\pi(u_2) \mid u \mid \underline{v_2})$; we can infer from the contrapositive of point 3 or point 4 of Lemma 44 respectively that $f(\pi(u_2) \cdot u \mid \underline{v_1}) \neq f(\pi(u_2) \cdot u \mid \underline{v_2})$. Since $f(u_1 \cdot u \mid \underline{v_1}) \neq f(u_1 \cdot u \mid \underline{v_2})$, we infer from the contrapositive of Lemma 46 that either $f(u_1 \mid u \cdot \underline{v_1}) \neq f(u_1 \mid u \cdot \underline{v_2})$ or $f(u_1 \mid u \mid \underline{v_1}) \neq f(u_1 \mid u \mid \underline{v_2})$.

Case 1: $f(u_1 \mid u \cdot \underline{v_1}) \neq f(u_1 \mid u \cdot \underline{v_2})$. If the number of left blocks in $f(u_1 \mid u \cdot \underline{v_1})$ is different from the number of left blocks in $f(u_1 \mid u \cdot \underline{v_2})$, then the number of left blocks in $f(\pi(u_2) \mid u \cdot \underline{v_1})$ is different from the number of left blocks in $f(\pi(u_2) \mid u \cdot \underline{v_2})$ and we are done. Suppose $f(u_1 \mid u \cdot \underline{v_1})$ and $f(u_1 \mid u \cdot \underline{v_2})$ have the same number of left blocks but the i^{th} left blocks are different. Suppose the i^{th} left block of $f(u_1 \mid u \cdot \underline{v_1})$ is $val_1(\chi_1)$ and the i^{th} left block of $f(u_1 \mid u \cdot \underline{v_2})$ is $val_1(\chi_2)$, where χ_1, χ_2 are some arrangements of some subsets $X_1, X_2 \subseteq X$ respectively. The i^{th} left block of $f(\pi(u_2) \mid u \cdot \underline{v_1})$ is $\pi(val_2)(\chi_1)$ and the i^{th} left block of $f(\pi(u_2) \mid u \cdot \underline{v_2})$ is $\pi(val_2)(\chi_2)$. Since $val_1(\chi_1) \neq val_1(\chi_2)$, we infer from condition 5 of Definition 36 that $\pi(val_2)(\chi_1) \neq \pi(val_2)(\chi_2)$. Hence, the i^{th} left blocks of $f(\pi(u_2) \mid u \cdot \underline{v_1})$ and $f(\pi(u_2) \mid u \cdot \underline{v_2})$ are different and we are done.

Case 2: $f(u_1 \mid u \mid \underline{v_1}) \neq f(u_1 \mid u \mid \underline{v_2})$. As we have seen in the second paragraph of this proof, $f_z(\pi(u_2) \mid u \cdot \underline{v_1}) = f_z(u_1 \mid u \cdot \underline{v_1})$ and $f_z(\pi(u_2) \mid u \cdot \underline{v_2}) = f_z(u_1 \mid u \cdot \underline{v_2})$. We infer from point 1 of Lemma 44 that $f_z(\pi(u_2) \mid u \mid \underline{v_1}) = f_z(u_1 \mid u \mid \underline{v_1})$ and $f_z(\pi(u_2) \mid u \mid \underline{v_2}) = f_z(u_1 \mid u \mid \underline{v_2})$. Since $f(u_1 \mid u \mid \underline{v_1}) \neq f(u_1 \mid u \mid \underline{v_2})$, $f_z(\pi(u_2) \mid u \mid \underline{v_1}) \neq f_z(\pi(u_2) \mid u \mid \underline{v_2})$, hence $f(\pi(u_2) \mid u \mid \underline{v_1}) \neq f(\pi(u_2) \mid u \mid \underline{v_2})$ and we are done. \blacktriangleleft

Proof of Lemma 38. Let $\delta_1 \delta_2 \dots$ be a sequence of data values. The equalizing scheme E is defined as follows: for every data word u , define $E(u)$ to be a permutation π such that $\pi(d_i) = \delta_i$ where d_i is the i^{th} f -influencing value in u . We infer from Lemma 40 that the i^{th} f -influencing value in $E(u)(u)$ is δ_i . For a set U of data words, we define the binary relation $\equiv_{U,f}^E$ as follows: $v_1 \equiv_{U,f}^E v_2$ if $f(E(u)(u) \mid \underline{v_1}) = f(E(u)(u) \mid \underline{v_2})$ for all $u \in U$. Note that $v_1 \equiv_f^E v_2$ iff $v_1 \equiv_{U,f}^E v_2$ for every $U \in (\Sigma \times D)^* / \equiv_S$. To prove the \equiv_f has finite index, it is sufficient to prove that $\equiv_{U,f}^E$ has finite index for every $U \in (\Sigma \times D)^* / \equiv_S$, since \equiv_S itself has finite index.

Next we will prove that $\equiv_{U,f}^E$ has finite index for every $U \in (\Sigma \times D)^* / \equiv_S$. Suppose $u_1 \equiv_S u_2$. Then $E(u_1)(u_1) \equiv_S E(u_2)(u_2)$. Suppose S reaches the configuration (q, val_1) after reading $E(u_1)(u_1)$ and reaches (q, val_2) after reading $E(u_2)(u_2)$. We infer from Lemma 50 that for any data word v and any i , if the i^{th} left block of $f(E(u_1)(u_1) \mid \underline{v})$ is $val_1(\chi)$, where χ is some arrangement of a subset $X' \subset X$, then the i^{th} left block of $f(E(u_2)(u_2) \mid \underline{v})$ is $val_2(\chi)$. Suppose v_1, v_2 are data words such that $f(E(u_1)(u_1) \mid \underline{v_1}) = f(E(u_1)(u_1) \mid \underline{v_2})$. For any i , the i^{th} left block of $f(E(u_1)(u_1) \mid \underline{v_1})$ is $val_1(\chi_1)$, where χ_1 is some arrangement of some subset $X_1 \subseteq X$ and the i^{th} left block of $f(E(u_1)(u_1) \mid \underline{v_2})$ is $val_1(\chi_2)$, where χ_2 is some arrangement of some subset $X_2 \subseteq X$. Since $f(E(u_1)(u_1) \mid \underline{v_1}) = f(E(u_1)(u_1) \mid \underline{v_2})$, $val_1(\chi_1) = val_1(\chi_2)$. For any i , the i^{th} left block of $f(E(u_2)(u_2) \mid \underline{v_1})$ is $val_2(\chi_1)$ and the i^{th} left block of $f(E(u_2)(u_2) \mid \underline{v_2})$ is $val_2(\chi_2)$. Since $val_1(\chi_1) = val_1(\chi_2)$, we infer from condition 5 of Definition 36 that $val_2(\chi_1) = val_2(\chi_2)$. Hence, $f(E(u_2)(u_2) \mid \underline{v_1}) = f(E(u_2)(u_2) \mid \underline{v_2})$. This

implies that $\equiv_{\{u_1\},f}^E$ and $\equiv_{\{u_2\},f}^E$ are the same whenever $u_1 \equiv_S u_2$. Hence, to prove that $\equiv_{U,f}^E$ has finite index for every $U \in (\Sigma \times D)^* / \equiv_S$, it is sufficient to prove that $\equiv_{\{u\},f}^E$ has finite index for some $u \in U$ for every $U \in (\Sigma \times D)^* / \equiv_S$.

Let u be an arbitrary data word. For any data word v , $f(E(u)(u) \mid v)$ has only data values from $E(u)(u)$ and has length bounded by a constant multiple of $|u|$. Hence, there are only finitely many possible distinct factored outputs $f(E(u)(u) \mid v)$ for all data words v . Hence, $\equiv_{\{u\},f}^E$ has finite index. This concludes the proof. \blacktriangleleft

E Proofs with Lengthy Case Analyses

► **Lemma 51.** *Suppose f is a transduction that is invariant under permutations and without data peeking, u is a data word and e is a data value. If d is a data value that is not f -influencing in u and d' is a safe replacement for d in u , then e is f -suffix influencing (resp. f -prefix influencing) in u iff e is f -suffix influencing (resp. f -prefix influencing) in $u[d/d']$.*

Proof. The idea for the proof is the following. If a data value e' and data word v certify that e is f -suffix influencing in u , then some permutations can be applied on e' and v to certify that e is f -suffix influencing in $u[d/d']$. Similar strategies work for the converse direction and for f -prefix influencing values.

Suppose $e = d$. We have to prove that d is not f -influencing in $u[d/d']$. Since d doesn't occur in $u[d/d']$, we get $u[d/d'][d/d''] = u[d/d']$ for any data value d'' . Hence, $f(u[d/d'][d/d''] \mid v) = f(u[d/d'] \mid v)$ for all data words v , so d is not f -suffix influencing in $u[d/d']$. Since d doesn't occur in $u[d/d']$, d is not f -prefix influencing in $u[d/d']$, as proved in Lemma 41.

Suppose $e \neq d$. First we will prove the statement about f -suffix influencing data values. First we will assume that e is f -suffix influencing in u and prove that e is f -suffix influencing in $u[d/d']$. There exists a safe replacement e' for e in u and a data word v such that

$$f(u[e/e'] \mid v) \neq f(u \mid v) \quad (1)$$

Let $e_1 \notin \text{data}(u \cdot v, *) \cup \{d, d', e, e'\}$ be a fresh data value and π_1 be the permutation that interchanges e' and e_1 and doesn't change any other data value. We apply π_1 to both sides of (1) to get $\pi_1(f(u[e/e'] \mid v)) \neq \pi_1(f(u \mid v))$. From Lemma 39, we then infer that

$$f(\pi_1(u[e/e']) \mid \pi_1(v)) \neq f(\pi_1(u) \mid \pi_1(v)) . \quad (2)$$

Since, e' is a safe replacement for e in u , e' doesn't occur in u . Hence, $\pi_1(u[e/e']) = u[e/e_1]$ and $\pi_1(u) = u$. Using these in (2), we get

$$f(u[e/e_1] \mid \pi_1(v)) \neq f(u \mid \pi_1(v)) . \quad (3)$$

Let π_2 be the permutation that interchanges d and d' and doesn't change any other data value. We apply π_2 to both sides of (3) to get $\pi_2(f(u[e/e_1] \mid \pi_1(v))) \neq \pi_2(f(u \mid \pi_1(v)))$. From Lemma 39, we then infer that $f(\pi_2(u[e/e_1]) \mid \pi_2(\pi_1(v))) \neq f(\pi_2(u) \mid \pi_2(\pi_1(v)))$. Since d' is a safe replacement for d in u , d' doesn't occur in u . By choice, $d' \neq e_1$. Hence, $\pi_2(u[e/e_1]) = u[e/e_1][d/d'] = u[d/d'][e/e_1]$ and $\pi_2(u) = u[d/d']$. Using these in the last inequality, we get $f(u[d/d'][e/e_1] \mid \pi_2(\pi_1(v))) \neq f(u[d/d'] \mid \pi_2(\pi_1(v)))$. This implies that e is f -suffix influencing in $u[d/d']$.

For the converse direction, we will first prove that d' is not f -suffix influencing in $u[d/d']$. Suppose for the sake of contradiction that d' is f -suffix influencing in $u[d/d']$. Then there

exists a data word v and a data value d'' that is a safe replacement for d' in $u[d/d']$ such that $f(u[d/d'] [d'/d''] \mid v) \neq f(u[d/d'] \mid v)$, so $f(u[d/d''] \mid v) \neq f(u[d/d'] \mid v)$. Now we apply the permutation π_3 that interchanges d and d'' on both sides of this inequality and Lemma 39 implies that $f(u \mid \pi_3(v)) \neq f(u[d/d'] \mid \pi_3(v))$. This shows that d is f -suffix influencing in u , a contradiction. Hence, d' is not f -suffix influencing in $u[d/d']$. Now, we have that d' is not f -suffix influencing in $u[d/d']$ and d is a safe replacement for d' in $u[d/d']$ and we have to prove that if e is f -suffix influencing in $u[d/d']$, then e is f -suffix influencing in u , which is same as $u[d/d'] [d'/d]$. This is similar to proving that if e is f -suffix influencing in u , then e is f -suffix influencing in $u[d/d']$, which we have already proved.

Next we will prove the statement about f -prefix influencing data values. We have already proved the statement for $e = d$, so assume that $e \neq d$. First assume that e doesn't occur in u . Then e is not f -prefix influencing in u . The value e is also not f -prefix influencing in $u[d/d']$ in the case where $d' \neq e$, since e doesn't occur in $u[d/d']$. We will prove that e is not f -prefix influencing in $u[d/e]$. Suppose for the sake of contradiction that e is f -prefix influencing in $u[d/e]$. There exist data words u', v such that e does not occur in u' and there exists a data value e' that is a safe replacement for e in $u[d/e] \cdot u' \cdot v$ such that $f(u[d/e] \cdot u' \mid v) \neq f(u[d/e] \cdot u' \mid v[e/e'])$. Now we apply the permutation π that interchanges d and e on both sides of this inequality and Lemma 39 implies that $f(u \cdot \pi(u') \mid \pi(v)) \neq f(u \cdot \pi(u') \mid \pi(v[e/e']))$. We have $\pi(v[e/e']) = \pi(v)[d/e']$, so $f(u \cdot \pi(u') \mid \pi(v)) \neq f(u \cdot \pi(u') \mid \pi(v)[d/e'])$. Since e doesn't occur in u' , d doesn't occur in $\pi(u')$. This implies that d is f -prefix influencing in u , a contradiction. So e is not f -prefix influencing in $u[d/e]$.

Next we will assume that e occurs in u . First we will assume that e is f -prefix influencing in u and prove that e is f -prefix influencing in $u[d/d']$. Suppose that e is f -prefix influencing in u . So there exist data words u', v such that e doesn't occur in u' and there exists a data value e' that is a safe replacement for e in $u \cdot u' \cdot v$ such that $f(u \cdot u' \mid v) \neq f(u \cdot u' \mid v[e/e'])$. Let $e_1 \notin \text{data}(u \cdot u' \cdot v, *) \cup \{d, d', e, e'\}$ be a fresh data value. The values e', e_1 don't occur in $u \cdot u' \cdot v$, so we can apply the permutation that interchanges e' and e_1 to both sides of the last inequality and Lemma 39 implies that $f(u \cdot u' \mid v) \neq f(u \cdot u' \mid v[e/e_1])$. Now we apply the permutation π that interchanges d and d' to both sides of the last inequality and from Lemma 39, we get that $f(u[d/d'] \cdot \pi(u') \mid \pi(v)) \neq f(u[d/d'] \cdot \pi(u') \mid \pi(v[e/e_1]))$. The value d' doesn't occur in u (since d' is a safe replacement for d in u) but e does, so $e \neq d'$. We also have $d \neq e$, $d \neq e_1$ and $d' \neq e_1$, so $\{d, d'\} \cap \{e, e_1\} = \emptyset$. Hence, $\pi(v[e/e_1]) = \pi(v)[e/e_1]$. So we get $f(u[d/d'] \cdot \pi(u') \mid \pi(v)) \neq f(u[d/d'] \cdot \pi(u') \mid \pi(v)[e/e_1])$, demonstrating that e is a f -prefix influencing value in $u[d/d']$ (note that since e doesn't occur in u' , it doesn't occur in $\pi(u')$ also). Hence we have shown that when $e \neq d$, if e is f -influencing in u , then e is f -influencing in $u[d/d']$.

For the converse direction, we will first prove that d' is not f -prefix influencing in $u[d/d']$. We have already proved that if e doesn't occur in u , then e is not f -prefix influencing in $u[d/e]$. Since d' doesn't occur in u , we can put $e = d'$ to conclude that d' is not f -prefix influencing in $u[d/d']$. Now, we have that d' is not f -prefix influencing in $u[d/d']$ and d is a safe replacement for d' in $u[d/d']$ and we have to prove that if e is f -prefix influencing in $u[d/d']$, then e is f -prefix influencing in u , which is same as $u[d/d'] [d'/d]$. This is similar to proving that if e is f -prefix influencing in u , then e is f -prefix influencing in $u[d/d']$. Hence the proof is complete. \blacktriangleleft

► **Lemma 52.** *Suppose f is a transduction that is invariant under permutations, $\sigma \in \Sigma$ is a letter and u is a data string. If d, e are data values, neither of which are f -influencing in u , then d is f -suffix influencing in $u \cdot (\sigma, d)$ iff e is f -suffix influencing in $u \cdot (\sigma, e)$. In addition, for any data value $\delta \notin \{d, e\}$, δ is f -suffix influencing in $u \cdot (\sigma, d)$ iff δ is f -suffix influencing*

in $u \cdot (\sigma, e)$.

Proof. We will assume that d is f -suffix influencing in $u \cdot (\sigma, d)$ and prove that e is f -suffix influencing in $u \cdot (\sigma, e)$. The proof of the other direction is similar. Let π be the permutation that interchanges d and e and doesn't change any other value. Since d is f -suffix influencing in $u \cdot (\sigma, d)$, there exist a data word v and a data value d' that is a safe replacement for d in $u \cdot (\sigma, d)$ satisfying the next inequality. Let π' be the permutation that interchanges d' and e and doesn't change any other value.

$$\begin{aligned}
f((u \cdot (\sigma, d))[d/d'] \mid v) &\neq f(u \cdot (\sigma, d) \mid v) && \text{[Definition 4]} \\
\pi(f((u \cdot (\sigma, d))[d/d'] \mid v)) &\neq \pi(f(u \cdot (\sigma, d) \mid v)) && \text{[apply } \pi \text{ to both sides]} \\
f(\pi((u \cdot (\sigma, d))[d/d'] \mid \pi(v))) &\neq f(\pi(u \cdot (\sigma, d)) \mid \pi(v)) && \text{[Lemma 39]} \quad (4) \\
f(\pi(u) \mid (\sigma, e) \cdot \pi(v)) &= f(u \mid (\sigma, e) \cdot \pi(v)) && \text{[Lemma 42]} \\
f(\pi(u) \cdot (\sigma, e) \mid \pi(v)) &= f(u \cdot (\sigma, e) \mid \pi(v)) && \text{[Lemma 44, point 2]} \\
f(\pi(u \cdot (\sigma, d)) \mid \pi(v)) &= f(u \cdot (\sigma, e) \mid \pi(v)) && (5) \\
f(u \mid (\sigma, d') \cdot \pi(v)) &= f(\pi(u) \mid (\sigma, d') \cdot \pi(v)) && \text{[Lemma 42]} \\
f(\pi'(u) \mid (\sigma, d') \cdot \pi(v)) &= f(\pi' \odot \pi(u) \mid (\sigma, d') \cdot \pi(v)) && \text{[Lemma 42]} \\
f(u[e/d'] \mid (\sigma, d') \cdot \pi(v)) &= f(\pi(u[d/d']) \mid (\sigma, d') \cdot \pi(v)) && [d' \notin \text{data}(u, *)] \\
f((u \cdot (\sigma, e))[e/d'] \mid \pi(v)) &= f(\pi((u \cdot (\sigma, d))[d/d'] \mid \pi(v))) && [d' \notin \text{data}(u, *)] \quad (6) \\
f((u \cdot (\sigma, e))[e/d'] \mid \pi(v)) &\neq f(u \cdot (\sigma, e) \mid \pi(v)) && [(4), (5), (6)]
\end{aligned}$$

From the last inequality above, we conclude that e is f -suffix influencing in $u \cdot (\sigma, e)$.

Next we will assume that δ is f -suffix influencing in $u \cdot (\sigma, d)$ and prove that δ is f -suffix influencing in $u \cdot (\sigma, e)$. The proof of the other direction is similar. Since δ is f -suffix influencing in $u \cdot (\sigma, d)$, there exists a data value δ' that is safe for replacing δ in $u \cdot (\sigma, d)$ and a data word v such that $f((u \cdot (\sigma, d))[\delta/\delta'] \mid v) \neq f(u \cdot (\sigma, d) \mid v)$. Let δ'' be a data value that is a safe replacement for δ in $u \cdot (\sigma, d) \cdot (\sigma, e)$. Let π_1 be the permutation that interchanges δ' and δ'' and doesn't change any other value. Let π_2 be the permutation that interchanges δ

and δ'' and doesn't change any other value.

$$\begin{aligned}
& f(\underline{(u \cdot (\sigma, d))[\delta/\delta']} \mid v) \neq f(\underline{u \cdot (\sigma, d)} \mid v) \\
& \pi_1(f(\underline{(u \cdot (\sigma, d))[\delta/\delta']} \mid v)) \neq \pi_1(f(\underline{u \cdot (\sigma, d)} \mid v)) \quad [\text{apply } \pi_1 \text{ on both sides}] \\
& f(\pi_1(\underline{(u \cdot (\sigma, d))[\delta/\delta']} \mid \pi_1(v))) \neq f(\pi_1(\underline{u \cdot (\sigma, d)} \mid \pi_1(v))) \quad [\text{Lemma 39}] \\
& f(\underline{(u \cdot (\sigma, d))[\delta/\delta'']} \mid \pi_1(v)) \neq f(\underline{u \cdot (\sigma, d)} \mid \pi_1(v)) \quad [\delta', \delta'' \notin \text{data}(u \cdot (\sigma, d), *)] \\
& \pi(f(\underline{(u \cdot (\sigma, d))[\delta/\delta'']} \mid \pi_1(v))) \neq \pi(f(\underline{u \cdot (\sigma, d)} \mid \pi_1(v))) \quad [\text{apply } \pi \text{ on both sides}] \\
& f(\pi(\underline{(u \cdot (\sigma, d))[\delta/\delta'']} \mid \pi_1(v))) \neq f(\pi(\underline{u \cdot (\sigma, d)} \mid \pi_1(v))) \quad [\text{Lemma 39}] \quad (7) \\
& f(\underline{u \mid (\sigma, d) \cdot \pi_1(v)}) = f(\underline{\pi(u)} \mid (\sigma, d) \cdot (\pi_1(v))) \quad [\text{Lemma 42}] \\
& \pi(f(\underline{u \mid (\sigma, d) \cdot \pi_1(v)})) = \pi(f(\underline{\pi(u)} \mid (\sigma, d) \cdot \pi_1(v))) \quad [\text{apply } \pi \text{ on both sides}] \\
& f(\pi(\underline{u \mid (\sigma, d) \cdot \pi_1(v)})) = f(\underline{\pi(u)} \mid (\sigma, d) \cdot \pi_1(v)) \quad [\text{Lemma 39}] \\
& f(\pi(\underline{u \mid (\sigma, d) \cdot \pi_1(v)})) = f(\underline{u \mid (\sigma, d) \cdot \pi_1(v)}) \quad [\text{Lemma 44, point 2}] \\
& f(\pi(\underline{u \mid (\sigma, d) \cdot \pi_1(v)})) = f(\underline{u \mid (\sigma, d) \cdot \pi_1(v)}) \quad [\text{Lemma 44, point 2}] \quad (8) \\
& d, e \notin \text{data}(\text{ifl}_f(\pi_2(u)), *) \quad [\{d, e\} \cap \{\delta, \delta''\} = \emptyset, \text{Lemma 40}] \\
& f(\pi_2(\underline{u \mid (\sigma, d) \cdot \pi_1(v)})) = f(\pi_2(\underline{\pi(u)} \mid (\sigma, d) \cdot \pi_1(v))) \quad [\text{Lemma 42}] \\
& \pi(f(\pi_2(\underline{u \mid (\sigma, d) \cdot \pi_1(v)}))) = \pi(f(\pi_2(\underline{\pi(u)} \mid (\sigma, d) \cdot \pi_1(v)))) \quad [\text{apply } \pi \text{ on both sides}] \\
& f(\pi(\pi_2(\underline{u \mid (\sigma, d) \cdot \pi_1(v)}))) = f(\pi_2(\underline{u \mid (\sigma, d) \cdot \pi_1(v)})) \quad [\text{Lemma 39}] \\
& f(\pi(\pi_2(\underline{u \mid (\sigma, d) \cdot \pi_1(v)}))) = f(\pi_2(\underline{u \mid (\sigma, d) \cdot \pi_1(v)})) \quad [\delta'' \notin \text{data}(u, *)] \\
& f(\pi(\pi_2(\underline{u \mid (\sigma, d) \cdot \pi_1(v)}))) = f(\pi_2(\underline{u \mid (\sigma, d) \cdot \pi_1(v)})) \quad [\text{Lemma 44, point 2}] \quad (9) \\
& f(\underline{u[\delta/\delta'']} \cdot (\sigma, e) \mid \pi \odot \pi_1(v)) \neq f(\underline{u \cdot (\sigma, e)} \mid \pi \odot \pi_1(v)) \quad [(7), (8), (9)] \\
& f(\underline{(u \cdot (\sigma, e))[\delta/\delta'']} \mid \pi \odot \pi_1(v)) \neq f(\underline{u \cdot (\sigma, e)} \mid \pi \odot \pi_1(v)) \quad [\delta \neq e]
\end{aligned}$$

The last inequality above certifies that δ is f -suffix influencing in $u \cdot (\sigma, e)$. \blacktriangleleft

► **Lemma 53.** Suppose f is a transduction that is invariant under permutations, $\sigma \in \Sigma$ is a letter and u is a data string. If d, e are data values, neither of which are f -influencing in u , then d is f -prefix influencing in $u \cdot (\sigma, d)$ iff e is f -prefix influencing in $u \cdot (\sigma, e)$. In addition, for any data value $\delta \notin \{d, e\}$, δ is f -prefix influencing in $u \cdot (\sigma, d)$ iff δ is f -prefix influencing in $u \cdot (\sigma, e)$.

Proof. We will assume that d is f -prefix influencing in $u \cdot (\sigma, d)$ and prove that e is f -prefix influencing in $u \cdot (\sigma, e)$. The proof of the other direction is similar. Let π be the permutation that interchanges d and e and doesn't change any other value. Since d is f -prefix influencing in $u \cdot (\sigma, d)$, we infer from Definition 4 that there exist data words u', v and a data value d' such that d doesn't occur in u' , d' is a safe replacement for d in $u \cdot (\sigma, d) \cdot u' \cdot v$ and $f(u \cdot (\sigma, d) \cdot u' \mid v[d/d']) \neq f(u \cdot (\sigma, d) \cdot u' \mid v)$. Applying the contrapositive of Lemma 46 to the above inequality, we infer that at least one of the following inequalities are true.

$$\begin{aligned}
& f(\underline{u \mid (\sigma, d) \cdot u' \mid v[d/d']}) \neq f(\underline{u \mid (\sigma, d) \cdot u' \mid v}) \\
& f(u \mid (\sigma, d) \cdot u' \cdot v[d/d']) \neq f(u \mid (\sigma, d) \cdot u' \cdot v)
\end{aligned}$$

Each of the above inequalities is taken up in one of the following cases. Let π be the permutation that interchanges d and e and doesn't change any other value. Let d'' be a

data value such that $d'' \notin \text{data}(u \cdot u' \cdot v, *) \cup \{d, e, d', \pi(d), \pi(d'), \pi(e), \pi(e')\}$. Let π' be the permutation that interchanges d' and d'' and doesn't change any other value.

Case 1:

$$\begin{aligned}
& f(\underline{u} \mid (\sigma, d) \cdot u' \mid v[d/d']) \neq f(\underline{u} \mid (\sigma, d) \cdot u' \mid \underline{v}) \\
& \pi'(f(\underline{u} \mid (\sigma, d) \cdot u' \mid v[d/d'])) \neq \pi'(f(\underline{u} \mid (\sigma, d) \cdot u' \mid \underline{v})) \quad [\text{apply } \pi' \text{ to both sides}] \\
& f(\underline{u} \mid (\sigma, d) \cdot u' \mid v[d/d'']) \neq f(\underline{u} \mid (\sigma, d) \cdot u' \mid \underline{v}) \quad [\text{Lemma 39, } d', d'' \notin \text{data}(u \cdot u' \cdot (\sigma, d) \cdot v, *)] \\
& \hspace{15em} (10) \\
& f(\underline{u} \mid (\sigma, d) \cdot u' \cdot v[d/d'']) = f(\pi(u) \mid (\sigma, d) \cdot u' \cdot v[d/d'']) \quad [\text{Lemma 42}] \\
& f(\underline{u} \mid (\sigma, d) \cdot u' \mid v[d/d'']) = f(\pi(u) \mid (\sigma, d) \cdot u' \mid v[d/d'']) \quad [\text{point 1 of Lemma 44}] \\
& \hspace{15em} (11) \\
& f(\underline{u} \mid (\sigma, d) \cdot u' \cdot v) = f(\pi(u) \mid (\sigma, d) \cdot u' \cdot v) \quad [\text{Lemma 42}] \\
& f(\underline{u} \mid (\sigma, d) \cdot u' \mid \underline{v}) = f(\pi(u) \mid (\sigma, d) \cdot u' \mid \underline{v}) \quad [\text{point 1 of Lemma 44}] \\
& \hspace{15em} (12)
\end{aligned}$$

$$\begin{aligned}
& f(\pi(u) \mid (\sigma, d) \cdot u' \mid v[d/d'']) \neq f(\pi(u) \mid (\sigma, d) \cdot u' \mid \underline{v}) \quad [(10), (11), (12)] \\
& \pi(f(\pi(u) \mid (\sigma, d) \cdot u' \mid v[d/d''])) \neq \pi(f(\pi(u) \mid (\sigma, d) \cdot u' \mid \underline{v})) \quad [\text{apply } \pi \text{ on both sides}] \\
& f(\underline{u} \mid (\sigma, e) \cdot \pi(u') \mid \pi(v)[e/d'']) \neq f(\underline{u} \mid (\sigma, e) \cdot \pi(u') \mid \pi(v)) \quad [\text{Lemma 39, } \pi(\pi(u)) = u, \pi(v[d/d'']) = \pi(v)[e/d'']] \\
& f(u \cdot (\sigma, e) \cdot \pi(u') \mid \pi(v)[e/d'']) \neq f(u \cdot (\sigma, e) \cdot \pi(u') \mid \pi(v)) \quad [\text{contrapositive of Lemma 44, point 4}]
\end{aligned}$$

Case 2:

$$\begin{aligned}
& f(u \mid (\sigma, d) \cdot u' \cdot v[d/d']) \neq f(u \mid (\sigma, d) \cdot u' \cdot v) \\
& \pi'(f(u \mid (\sigma, d) \cdot u' \cdot v[d/d'])) \neq \pi'(f(u \mid (\sigma, d) \cdot u' \cdot v)) \quad [\text{apply } \pi' \text{ on both sides}] \\
& f(u \mid (\sigma, d) \cdot u' \cdot v[d/d'']) \neq f(u \mid (\sigma, d) \cdot u' \cdot v) \quad [\text{Lemma 39, } d', d'' \notin \text{data}(u \cdot u' \cdot (\sigma, d) \cdot v, *)] \\
& f(u \mid \pi((\sigma, d) \cdot u' \cdot v[d/d''])) \neq f(u \mid \pi((\sigma, d) \cdot u' \cdot v)) \quad [\text{Lemma 43}] \\
& f(u \mid (\sigma, e) \cdot \pi(u') \cdot \pi(v)[e/d'']) \neq f(u \mid (\sigma, e) \cdot \pi(u') \cdot \pi(v)) \\
& f(u \cdot (\sigma, e) \cdot \pi(u') \mid \pi(v)[e/d'']) \neq f(u \cdot (\sigma, e) \cdot \pi(u') \mid \pi(v)) \quad [\text{contrapositive of Lemma 44, point 3}]
\end{aligned}$$

Since d doesn't occur in u' , e doesn't occur in $\pi(u')$. The last inequalities in each of the above cases certify that e is f -prefix influencing in $u \cdot (\sigma, e)$.

Next we will assume that δ is f -prefix influencing in $u \cdot (\sigma, d)$ and prove that δ is f -prefix influencing in $u \cdot (\sigma, e)$. The proof of the other direction is similar. Since δ is f -prefix influencing in $u \cdot (\sigma, d)$, we infer from Definition 4 that there exist data words u', v and a data value δ' such that δ doesn't occur in u' , δ' is a safe replacement for δ in $u \cdot (\sigma, d) \cdot u' \cdot v$ and $f(u \cdot (\sigma, d) \cdot u' \mid v[\delta/\delta']) \neq f(u \cdot (\sigma, d) \cdot u' \mid \underline{v})$. Applying the contrapositive of Lemma 46 to the above inequality, we infer that at least one of the following inequalities are true.

$$\begin{aligned}
& f(\underline{u} \mid (\sigma, d) \cdot u' \mid v[\delta/\delta']) \neq f(\underline{u} \mid (\sigma, d) \cdot u' \mid \underline{v}) \\
& f(u \mid (\sigma, d) \cdot u' \cdot v[\delta/\delta']) \neq f(u \mid (\sigma, d) \cdot u' \cdot v)
\end{aligned}$$

Each of the above inequalities is taken up in one of the following cases. Let π be the permutation that interchanges d and e and doesn't change any other value. Let δ'' be a data value such that $\delta'' \notin \text{data}(u \cdot u' \cdot v, *) \cup \{d, e, \delta'\}$. Let π' be the permutation that interchanges δ' and δ'' and doesn't change any other value.

Case 1:

$$\begin{aligned}
& f(\underline{u} \mid (\sigma, d) \cdot u' \mid \underline{v[\delta/\delta']}) \neq f(\underline{u} \mid (\sigma, d) \cdot u' \mid \underline{v}) \\
& \pi'(f(\underline{u} \mid (\sigma, d) \cdot u' \mid \underline{v[\delta/\delta']})) \neq \pi'(f(\underline{u} \mid (\sigma, d) \cdot u' \mid \underline{v})) \quad [\text{apply } \pi' \text{ to both sides}] \\
& f(\underline{u} \mid (\sigma, d) \cdot u' \mid \underline{v[\delta/\delta'']}) \neq f(\underline{u} \mid (\sigma, d) \cdot u' \mid \underline{v}) \quad [\text{Lemma 39, } \delta', \delta'' \notin \mathbf{data}(u \cdot u' \cdot (\sigma, d) \cdot v, *)] \\
& \quad (13) \\
& f(\underline{u} \mid (\sigma, d) \cdot u' \cdot v[\delta/\delta'']) = f(\pi(u) \mid (\sigma, d) \cdot u' \cdot v[\delta/\delta'']) \quad [\text{Lemma 42}] \\
& f(\underline{u} \mid (\sigma, d) \cdot u' \mid \underline{v[\delta/\delta'']}) = f(\pi(u) \mid (\sigma, d) \cdot u' \mid \underline{v[\delta/\delta'']}) \quad [\text{point 1 of Lemma 44}] \\
& \quad (14) \\
& f(\underline{u} \mid (\sigma, d) \cdot u' \cdot v) = f(\pi(u) \mid (\sigma, d) \cdot u' \cdot v) \quad [\text{Lemma 42}] \\
& f(\underline{u} \mid (\sigma, d) \cdot u' \mid \underline{v}) = f(\pi(u) \mid (\sigma, d) \cdot u' \mid \underline{v}) \quad [\text{point 1 of Lemma 44}] \\
& \quad (15) \\
& f(\pi(u) \mid (\sigma, d) \cdot u' \mid \underline{v[\delta/\delta'']}) \neq f(\pi(u) \mid (\sigma, d) \cdot u' \mid \underline{v}) \quad [(13), (14), (15)] \\
& \pi(f(\pi(u) \mid (\sigma, d) \cdot u' \mid \underline{v[\delta/\delta'']})) \neq \pi(f(\pi(u) \mid (\sigma, d) \cdot u' \mid \underline{v})) \quad [\text{apply } \pi \text{ on both sides}] \\
& f(\underline{u} \mid (\sigma, e) \cdot \pi(u') \mid \underline{\pi(v)[\delta/\delta'']}) \neq f(\underline{u} \mid (\sigma, e) \cdot \pi(u') \mid \underline{\pi(v)}) \quad [\text{Lemma 39, } \pi(\pi(u)) = u, \{d, e\} \cap \{\delta, \delta''\} = \emptyset] \\
& f(u \cdot (\sigma, e) \cdot \pi(u') \mid \underline{\pi(v)[\delta/\delta'']}) \neq f(u \cdot (\sigma, e) \cdot \pi(u') \mid \underline{\pi(v)}) \quad [\text{contrapositive of Lemma 44, point 4}]
\end{aligned}$$

Case 2:

$$\begin{aligned}
& f(u \mid (\sigma, d) \cdot u' \cdot v[\delta/\delta']) \neq f(u \mid (\sigma, d) \cdot u' \cdot v) \\
& \pi'(f(u \mid (\sigma, d) \cdot u' \cdot v[\delta/\delta'])) \neq \pi'(f(u \mid (\sigma, d) \cdot u' \cdot v)) \quad [\text{apply } \pi' \text{ on both sides}] \\
& f(u \mid (\sigma, d) \cdot u' \cdot v[\delta/\delta'']) \neq f(u \mid (\sigma, d) \cdot u' \cdot v) \quad [\text{Lemma 39, } \delta', \delta'' \notin \mathbf{data}(u \cdot u' \cdot (\sigma, d) \cdot v, *)] \\
& f(u \mid \pi((\sigma, d) \cdot u' \cdot v[\delta/\delta''])) \neq f(u \mid \pi((\sigma, d) \cdot u' \cdot v)) \quad [\text{Lemma 43}] \\
& f(u \mid (\sigma, e) \cdot \pi(u') \cdot \pi(v)[\delta/\delta'']) \neq f(u \mid (\sigma, e) \cdot \pi(u') \cdot \pi(v)) \\
& f(u \cdot (\sigma, e) \cdot \pi(u') \mid \underline{\pi(v)[\delta/\delta'']}) \neq f(u \cdot (\sigma, e) \cdot \pi(u') \mid \underline{\pi(v)}) \quad [\text{contrapositive of Lemma 44, point 3}]
\end{aligned}$$

Since δ doesn't occur in u' , δ doesn't occur in $\pi(u')$. The last inequalities in each of the above cases certify that δ is f -prefix influencing in $u \cdot (\sigma, e)$. \blacktriangleleft

Proof of Lemma 14. Since $u_1 \equiv_f u_2$, there exists a permutation π satisfying the conditions of Definition 7. Let $z = |u_1| - |u_2|$.

Proof of 1. Suppose d_1^i is f -suffix influencing in $u_1 \cdot (\sigma, d_1^j)$. There exist a data word v and a safe replacement d' for d_1^j in $u_1 \cdot (\sigma, d_1^j)$ such that $f((u_1 \cdot (\sigma, d_1^j))[d_1^i/d'] \mid v) \neq f(u_1 \cdot (\sigma, d_1^j) \mid v)$. Let d'' be a data value that is a safe replacement for d_1^i in $u_1 \cdot (\sigma, d_1^j) \cdot v \cdot \pi(u_2)$. Let π_1 be the permutation that interchanges d' and d'' and doesn't change any other value. Let π_2

be the permutation that interchanges d_1^i and d'' and doesn't change any other value.

$$\begin{aligned}
& f((u_1 \cdot (\sigma, d_1^j))[d_1^i/d'] \mid v) \neq f(u_1 \cdot (\sigma, d_1^j) \mid v) && \text{[Definition 4]} \\
& \pi_1(f((u_1 \cdot (\sigma, d_1^j))[d_1^i/d'] \mid v)) \neq \pi_1(f(u_1 \cdot (\sigma, d_1^j) \mid v)) && \text{[apply } \pi_1 \text{ on both sides]} \\
& f(\pi_1((u_1 \cdot (\sigma, d_1^j))[d_1^i/d'] \mid \pi_1(v))) \neq f(\pi_1(u_1 \cdot (\sigma, d_1^j) \mid \pi_1(v))) && \text{[Lemma 39]} \\
& f((u_1 \cdot (\sigma, d_1^j))[d_1^i/d''] \mid \pi_1(v)) \neq f(u_1 \cdot (\sigma, d_1^j) \mid \pi_1(v)) && [\{d', d''\} \notin \text{data}(u_1 \cdot (\sigma, d_1^j), *)] \\
& && (16) \\
& f(u_1 \mid (\sigma, d_1^j) \cdot \pi_1(v)) = f_z(\pi(u_2) \mid (\sigma, d_1^j) \cdot \pi_1(v)) && \text{[Definition 7]} \\
& f(u_1 \cdot (\sigma, d_1^j) \mid \pi_1(v)) = f_z(\pi(u_2) \cdot (\sigma, d_1^j) \mid \pi_1(v)) && \text{[Lemma 44, point 2]} \\
& && (17) \\
& f(u_1 \mid (\sigma, d_1^j) \cdot \pi_2^{-1} \odot \pi_1(v)) = f_z(\pi(u_2) \mid (\sigma, d_1^j) \cdot \pi_2^{-1} \odot \pi_1(v)) && \text{[Definition 7]} \\
& \pi_2(f(u_1 \mid (\sigma, d_1^j) \cdot \pi_2^{-1} \odot \pi_1(v))) = \pi_2(f_z(\pi(u_2) \mid (\sigma, d_1^j) \cdot \pi_2^{-1} \odot \pi_1(v))) && \text{[apply } \pi_2 \text{ on both sides]} \\
& f(\pi_2(u_1) \mid \pi_2((\sigma, d_1^j) \cdot \pi_2^{-1} \odot \pi_1(v))) = f_z(\pi_2(\pi(u_2)) \mid \pi_2((\sigma, d_1^j) \cdot \pi_2^{-1} \odot \pi_1(v))) && \text{[Lemma 39]} \\
& f(u_1[d_1^i/d''] \mid (\sigma, d_1^j)[d_1^i/d''] \cdot \pi_1(v)) = f_z(\pi(u_2)[d_1^i/d''] \mid (\sigma, d_1^j)[d_1^i/d''] \cdot \pi_1(v)) && [d'' \notin \text{data}(u_1 \cdot (\sigma, d_1^j) \cdot \pi(u_2), *)] \\
& f((u_1 \cdot (\sigma, d_1^j))[d_1^i/d''] \mid \pi_1(v)) = f_z((\pi(u_2) \cdot (\sigma, d_1^j))[d_1^i/d''] \mid \pi_1(v)) && \text{[Lemma 44, point 2]} \\
& && (18) \\
& f_z((\pi(u_2) \cdot (\sigma, d_1^j))[d_1^i/d''] \mid \pi_1(v)) \neq f_z(\pi(u_2) \cdot (\sigma, d_1^j) \mid \pi_1(v)) && [(16), (17), (18)] \\
& f((\pi(u_2) \cdot (\sigma, d_1^j))[d_1^i/d''] \mid \pi_1(v)) \neq f(\pi(u_2) \cdot (\sigma, d_1^j) \mid \pi_1(v))
\end{aligned}$$

Since d'' is a safe replacement for d_1^i in $\pi(u_2) \cdot (\sigma, d_1^j)$, the last inequality above certifies that d_1^i is f -suffix influencing in $\pi(u_2) \cdot (\sigma, d_1^j)$. Since $\pi(u_2) \cdot (\sigma, d_1^j) = \pi(u_2 \cdot (\sigma, \pi^{-1}(d_1^j)))$, we infer that d_1^i is f -suffix influencing in $\pi(u_2 \cdot (\sigma, \pi^{-1}(d_1^j)))$. From Lemma 40, we infer that $\pi^{-1}(d_1^i)$ is f -suffix influencing in $u_2 \cdot (\sigma, \pi^{-1}(d_1^j))$.

Case 1: $(d_1^j, d_2^j) \in \{(d_1^k, d_2^k) \mid 1 \leq k \leq m\}$. In this case, $\pi^{-1}(d_1^j) = d_2^j$. So $\pi^{-1}(d_1^i)$ is f -suffix influencing in $u_2 \cdot (\sigma, d_2^j)$. Since d_1^i is f -suffix influencing in $u_1 \cdot (\sigma, d_1^j)$, we infer from Lemma 13 that d_1^i is f -suffix influencing in u_1 or $d_1^i = d_1^j$. Either way, $d_1^i \in \{d_1^k \mid 1 \leq k \leq m\}$, so $(d_1^i, d_2^i) \in \{(d_1^k, d_2^k) \mid 1 \leq k \leq m\}$. Hence $\pi^{-1}(d_1^i) = d_2^i$, so d_2^i is f -suffix influencing in $u_2 \cdot (\sigma, d_2^j)$.

Case 2: $(d_1^j, d_2^j) = (d_1^0, d_2^0)$. Since $d_1^j = d_1^0$ is not f -influencing in u_1 , $\pi^{-1}(d_1^j)$ is not f -influencing in u_2 . From the hypothesis of this lemma, $d_2^j = d_2^0$ is not f -influencing in u_2 . If $(d_1^i, d_2^i) \in \{(d_1^k, d_2^k) \mid 1 \leq k \leq m\}$, then $\pi^{-1}(d_1^i) = d_2^i$. So d_2^i is f -suffix influencing in $u_2 \cdot (\sigma, \pi^{-1}(d_1^j))$. From Lemma 52, we conclude that d_2^i is f -suffix influencing in $u_2 \cdot (\sigma, d_2^j)$. The other possibility is that $(d_1^i, d_2^i) = (d_1^0, d_2^0) = (d_1^j, d_2^j)$. Since $d_1^i = d_1^0$ is not f -influencing in u_1 , $\pi^{-1}(d_1^i)$ is not f -influencing in u_2 . Since $\pi^{-1}(d_1^i) = \pi^{-1}(d_1^j)$ is f -suffix influencing in $u_2 \cdot (\sigma, \pi^{-1}(d_1^j))$, from Lemma 52, we conclude that $d_2^i = d_2^j$ is f -suffix influencing in $u_2 \cdot (\sigma, d_2^j)$. If d_2^i is f -suffix influencing in $u_2 \cdot (\sigma, d_2^j)$, we can prove that d_1^i is f -suffix influencing in $u_1 \cdot (\sigma, d_1^j)$ with a similar proof.

Suppose d_1^i is f -prefix influencing in $u_1 \cdot (\sigma, d_1^j)$. We infer from Definition 4 that there exist data words u', v and a data value d' such that d_1^i doesn't occur in u' , d' is a safe replacement for d_1^i in $u_1 \cdot (\sigma, d_1^j) \cdot u' \cdot v$ and $f(u_1 \cdot (\sigma, d_1^j) \cdot u' \mid \underline{v[d_1^i/d']}) \neq f(u_1 \cdot (\sigma, d_1^j) \cdot u' \mid \underline{v})$. Let d'' be a data value that is a safe replacement for d_1^i in $u_1 \cdot (\sigma, d_1^j) \cdot u' \cdot v \cdot \pi(u_2)$. Let π_1 be the permutation that interchanges d' and d'' and doesn't change any other value.

$$\begin{aligned}
& f(u_1 \cdot (\sigma, d_1^j) \cdot u' \mid \underline{v[d_1^i/d']}) \neq f(u_1 \cdot (\sigma, d_1^j) \cdot u' \mid \underline{v}) \\
& \pi_1(f(u_1 \cdot (\sigma, d_1^j) \cdot u' \mid \underline{v[d_1^i/d']})) \neq \pi_1(f(u_1 \cdot (\sigma, d_1^j) \cdot u' \mid \underline{v})) \quad [\text{apply } \pi_1 \text{ on both sides}] \\
& f(u_1 \cdot (\sigma, d_1^j) \cdot u' \mid \underline{v[d_1^i/d'']}) \neq f(u_1 \cdot (\sigma, d_1^j) \cdot u' \mid \underline{v}) \quad [\text{Lemma 39, } d', d'' \notin \text{data}(u_1 \cdot (\sigma, d_1^j) \cdot u' \cdot v, *)] \\
& f(\pi(u_2) \cdot (\sigma, d_1^j) \cdot u' \mid \underline{v[d_1^i/d'']}) \neq f(\pi(u_2) \cdot (\sigma, d_1^j) \cdot u' \mid \underline{v}) \quad [\text{last condition on } \pi \text{ in Definition 7}]
\end{aligned}$$

The last inequality above implies that d_1^i is f -prefix influencing in $\pi(u_2) \cdot (\sigma, d_1^j)$. Since, $\pi(u_2) \cdot (\sigma, d_1^j) = \pi(u_2 \cdot (\sigma, \pi^{-1}(d_1^j)))$, d_1^i is f -prefix influencing in $\pi(u_2 \cdot (\sigma, \pi^{-1}(d_1^j)))$. From Lemma 40, we infer that $\pi^{-1}(d_1^i)$ is f -prefix influencing in $u_2 \cdot (\sigma, \pi^{-1}(d_1^j))$.

Case 1: $(d_1^j, d_2^j) \in \{(d_1^k, d_2^k) \mid 1 \leq k \leq m\}$. In this case, $\pi^{-1}(d_1^j) = d_2^j$ (since π maps $\text{ifl}_f(u_2)$ to $\text{ifl}_f(u_1)$), so $\pi^{-1}(d_1^i)$ is f -prefix influencing in $u_2 \cdot (\sigma, d_2^j)$. Since d_1^i is f -prefix influencing in $u_1 \cdot (\sigma, d_1^j)$, we infer from Lemma 13 that d_1^i is f -prefix influencing in u_1 or $d_1^i = d_1^j$. Either way, $d_1^i \in \{d_1^k \mid 1 \leq k \leq m\}$, so $(d_1^i, d_2^i) \in \{(d_1^k, d_2^k) \mid 1 \leq k \leq m\}$. Hence, $\pi^{-1}(d_1^i) = d_2^i$, so d_2^i is f -prefix influencing in $u_2 \cdot (\sigma, d_2^j)$.

Case 2: $(d_1^j, d_2^j) = (d_1^0, d_2^0)$. In this case, $d_2^j = d_2^0$ is not f -influencing in u_2 , and $\pi^{-1}(d_1^j) = \pi^{-1}(d_1^0)$ is not f -influencing in u_2 (since d_1^0 is not f -influencing in u_1). If $(d_1^i, d_2^i) \in \{(d_1^k, d_2^k) \mid 1 \leq k \leq m\}$, then $\pi^{-1}(d_1^i) = d_2^i$. So d_2^i is f -prefix influencing in $u_2 \cdot (\sigma, \pi^{-1}(d_1^j))$. From Lemma 53, we infer that d_2^i is f -prefix influencing in $u_2 \cdot (\sigma, d_2^j)$. The other possibility is that $(d_1^i, d_2^i) = (d_1^0, d_2^0) = (d_1^j, d_2^j)$. Since $d_1^i = d_1^0$ is not f -influencing in u_1 , $\pi^{-1}(d_1^i)$ is not f -influencing in u_2 . Since $\pi^{-1}(d_1^i) = \pi^{-1}(d_1^j)$ is f -prefix influencing in $u_2 \cdot (\sigma, \pi^{-1}(d_1^j))$, from Lemma 53, we conclude that $d_2^i = d_2^j$ is f -prefix influencing in $u_2 \cdot (\sigma, d_2^j)$. If d_2^i is f -prefix influencing in $u_2 \cdot (\sigma, d_2^j)$, we can prove that d_1^i is f -prefix influencing in $u_1 \cdot (\sigma, d_1^j)$ with a similar proof.

Proof of 2. Let π' be the permutation that interchanges d_1^0 and $\pi(d_2^0)$ and doesn't change any other value. To prove that $u_1 \cdot (\sigma, d_1^j) \equiv_f u_2 \cdot (\sigma, d_2^j)$, we will prove that the permutation $\pi' \odot \pi$ satisfies all the conditions of Definition 7. Note that $\pi' \odot \pi(d_2^j) = d_1^j$. From Lemma 13, we infer that f -influencing values in $u_1 \cdot (\sigma, d_1^j)$ are among $\{d_1^k \mid 1 \leq k \leq m\} \cup \{d_1^j\}$ and that f -influencing values in $u_2 \cdot (\sigma, d_2^j)$ are among $\{d_2^k \mid 1 \leq k \leq m\} \cup \{d_2^j\}$. We infer from point 1 of this lemma that d_1^j is f -suffix influencing (resp. f -prefix influencing) in $u_1 \cdot (\sigma, d_1^j)$ iff d_2^j is f -suffix influencing (resp. f -prefix influencing) in $u_2 \cdot (\sigma, d_2^j)$. We also infer from point 1 of this lemma that for $(d_1^i, d_2^i) \in \{(d_1^k, d_2^k) \mid 1 \leq k \leq m\}$, d_1^i is f -suffix influencing (resp. f -prefix influencing) in $u_1 \cdot (\sigma, d_1^j)$ iff d_2^i is f -suffix influencing (resp. f -prefix influencing) in $u_2 \cdot (\sigma, d_2^j)$. Since, $\pi' \odot \pi(d_2^i) = d_1^i$ and $\pi' \odot \pi(d_2^j) = d_1^j$, we infer that $\text{aifl}_f(\pi' \odot \pi(u_2 \cdot (\sigma, d_2^j))) = \text{aifl}_f(u_1 \cdot (\sigma, d_1^j))$.

Let v be an arbitrary data word. Since d_2^0 is not f -influencing in u_2 , $\pi(d_2^0)$ is not f -influencing in u_1 .

$$\begin{aligned}
& f_z(\pi(u_2) \mid (\sigma, d_1^j) \cdot v) = f(\underline{u_1} \mid (\sigma, d_1^j) \cdot v) \quad [\text{first condition on } \pi \text{ in Definition 7}] \\
& f_z(\pi' \odot \pi(u_2) \mid (\sigma, d_1^j) \cdot v) = f(\underline{u_1} \mid (\sigma, d_1^j) \cdot v) \quad [\text{Lemma 42}] \\
& f_z(\pi' \odot \pi(u_2) \cdot (\sigma, d_1^j) \mid v) = f(\underline{u_1} \cdot (\sigma, d_1^j) \mid v) \quad [\text{Lemma 44, point 2}] \\
& f_z(\pi' \odot \pi(u_2 \cdot (\sigma, d_2^j)) \mid v) = f(\underline{u_1} \cdot (\sigma, d_1^j) \mid v)
\end{aligned}$$

Since the last inequality above holds for any data word v , it proves the first condition of Definition 7.

For the last condition of Definition 7, suppose u, v_1, v_2 are arbitrary data values and $f(u_1 \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_1}) = f(u_1 \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_2})$. Since, $u_1 \equiv_f u_2$ and π satisfies all the conditions

of Definition 7, we infer that $f(\pi(u_2) \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_1}) = f(\pi(u_2) \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_2})$.

$$\begin{aligned}
& f(\pi(u_2) \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_1}) = f(\pi(u_2) \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_2}) \\
& f(\pi(u_2) \mid (\sigma, d_1^j) \cdot u \cdot v_1) = f(\pi(u_2) \mid (\sigma, d_1^j) \cdot u \cdot v_2) \quad [\text{Lemma 44, point 3}] \\
& f(\pi(u_2) \mid \pi'((\sigma, d_1^j) \cdot u \cdot v_1)) = f(\pi(u_2) \mid \pi'((\sigma, d_1^j) \cdot u \cdot v_2)) \quad [\text{Lemma 43, } \pi(d_2^0), d_1^0 \notin \text{data}(\text{aifl}_f(\pi(u_2), *)) \\
& \pi'(f(\pi(u_2) \mid \pi'((\sigma, d_1^j) \cdot u \cdot v_1))) = \pi'(f(\pi(u_2) \mid \pi'((\sigma, d_1^j) \cdot u \cdot v_2))) \quad [\text{apply } \pi' \text{ on both sides}] \\
& f(\pi' \odot \pi(u_2) \mid (\sigma, d_1^j) \cdot u \cdot v_1) = f(\pi' \odot \pi(u_2) \mid (\sigma, d_1^j) \cdot u \cdot v_2) \quad [\text{Lemma 39}] \quad (19) \\
& f(\pi(u_2) \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_1}) = f(\pi(u_2) \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_2}) \\
& f(\pi(u_2) \mid (\sigma, d_1^j) \cdot u \mid \underline{v_1}) = f(\pi(u_2) \mid (\sigma, d_1^j) \cdot u \mid \underline{v_2}) \quad [\text{Lemma 44, point 4}] \\
& \quad \quad \quad (20) \\
& f(\pi' \odot \pi(u_2) \mid (\sigma, d_1^j) \cdot u \cdot v_1) = f(\pi(u_2) \mid (\sigma, d_1^j) \cdot u \cdot v_1) \quad [\text{Lemma 42, } \pi(d_2^0), d_1^0 \notin \text{data}(\text{aifl}_f(\pi(u_2), *)) \\
& f(\pi' \odot \pi(u_2) \mid (\sigma, d_1^j) \cdot u \mid \underline{v_1}) = f(\pi(u_2) \mid (\sigma, d_1^j) \cdot u \mid \underline{v_1}) \quad [\text{Lemma 44, point 1}] \\
& \quad \quad \quad (21) \\
& f(\pi' \odot \pi(u_2) \mid (\sigma, d_1^j) \cdot u \cdot v_2) = f(\pi(u_2) \mid (\sigma, d_1^j) \cdot u \cdot v_2) \quad [\text{Lemma 42, } \pi(d_2^0), d_1^0 \notin \text{data}(\text{aifl}_f(\pi(u_2), *)) \\
& f(\pi' \odot \pi(u_2) \mid (\sigma, d_1^j) \cdot u \mid \underline{v_2}) = f(\pi(u_2) \mid (\sigma, d_1^j) \cdot u \mid \underline{v_2}) \quad [\text{Lemma 44, point 1}] \\
& \quad \quad \quad (22) \\
& f(\pi' \odot \pi(u_2) \mid (\sigma, d_1^j) \cdot u \mid \underline{v_1}) = f(\pi' \odot \pi(u_2) \mid (\sigma, d_1^j) \cdot u \mid \underline{v_2}) \quad [(20), (21), (22)] \quad (23) \\
& f(\pi' \odot \pi(u_2) \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_1}) = f(\pi' \odot \pi(u_2) \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_2}) \quad [(19), (23), \text{Lemma 46}] \\
& f(\pi' \odot \pi(u_2 \cdot (\sigma, d_2^j)) \cdot u \mid \underline{v_1}) = f(\pi' \odot \pi(u_2 \cdot (\sigma, d_2^j)) \cdot u \mid \underline{v_2})
\end{aligned}$$

Hence, if $f(u_1 \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_1}) = f(u_1 \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_2})$, then $f(\pi' \odot \pi(u_2 \cdot (\sigma, d_2^j)) \cdot u \mid \underline{v_1}) = f(\pi' \odot \pi(u_2 \cdot (\sigma, d_2^j)) \cdot u \mid \underline{v_2})$.

Conversely, suppose $f(\pi' \odot \pi(u_2 \cdot (\sigma, d_2^j)) \cdot u \mid \underline{v_1}) = f(\pi' \odot \pi(u_2 \cdot (\sigma, d_2^j)) \cdot u \mid \underline{v_2})$. Then we have $f(\pi' \odot \pi(u_2) \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_1}) = f(\pi' \odot \pi(u_2) \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_2})$. Recall that $\pi(d_2^0)$ and

d_1^0 are not f -influencing in $\pi(u_2)$.

$$\begin{aligned}
& f(\pi' \odot \pi(u_2) \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_1}) = f(\pi' \odot \pi(u_2) \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_2}) \\
& f(\pi' \odot \pi(u_2) \mid (\sigma, d_1^j) \cdot u \cdot \underline{v_1}) = f(\pi' \odot \pi(u_2) \mid (\sigma, d_1^j) \cdot u \cdot \underline{v_2}) \quad [\text{Lemma 44, point 3}] \\
& f(\pi' \odot \pi(u_2) \mid \underline{\pi'((\sigma, d_1^j) \cdot u \cdot v_1)}) = f(\pi' \odot \pi(u_2) \mid \underline{\pi'((\sigma, d_1^j) \cdot u \cdot v_2)}) \quad [\text{Lemma 43, } \pi(d_2^0), d_1^0 \notin \text{data}(\text{aifl}_f(\pi' \cdot \pi(u_2), *)))] \\
& \pi'(f(\pi' \odot \pi(u_2) \mid \underline{\pi'((\sigma, d_1^j) \cdot u \cdot v_1)})) = \pi'(f(\pi' \odot \pi(u_2) \mid \underline{\pi'((\sigma, d_1^j) \cdot u \cdot v_2)})) \quad [\text{apply } \pi' \text{ on both sides}] \\
& f(\pi(u_2) \mid (\sigma, d_1^j) \cdot u \cdot \underline{v_1}) = f(\pi(u_2) \mid (\sigma, d_1^j) \cdot u \cdot \underline{v_2}) \quad [\text{Lemma 39}] \\
& \quad \quad \quad (24) \\
& f(\pi' \odot \pi(u_2) \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_1}) = f(\pi' \odot \pi(u_2) \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_2}) \\
& f(\pi' \odot \pi(u_2) \mid (\sigma, d_1^j) \cdot u \mid \underline{v_1}) = f(\pi' \odot \pi(u_2) \mid (\sigma, d_1^j) \cdot u \mid \underline{v_2}) \quad [\text{Lemma 44, point 4}] \\
& \quad \quad \quad (25) \\
& f(\pi' \odot \pi(u_2) \mid (\sigma, d_1^j) \cdot u \cdot \underline{v_1}) = f(\pi(u_2) \mid (\sigma, d_1^j) \cdot u \cdot \underline{v_1}) \quad [\text{Lemma 42, } \pi(d_2^0), d_1^0 \notin \text{data}(\text{aifl}_f(\pi(u_2), *)))] \\
& f(\pi' \odot \pi(u_2) \mid (\sigma, d_1^j) \cdot u \mid \underline{v_1}) = f(\pi(u_2) \mid (\sigma, d_1^j) \cdot u \mid \underline{v_1}) \quad [\text{Lemma 44, point 1}] \\
& \quad \quad \quad (26) \\
& f(\pi' \odot \pi(u_2) \mid (\sigma, d_1^j) \cdot u \cdot \underline{v_2}) = f(\pi(u_2) \mid (\sigma, d_1^j) \cdot u \cdot \underline{v_2}) \quad [\text{Lemma 42, } \pi(d_2^0), d_1^0 \notin \text{data}(\text{aifl}_f(\pi(u_2), *)))] \\
& f(\pi' \odot \pi(u_2) \mid (\sigma, d_1^j) \cdot u \mid \underline{v_2}) = f(\pi(u_2) \mid (\sigma, d_1^j) \cdot u \mid \underline{v_2}) \quad [\text{Lemma 44, point 1}] \\
& \quad \quad \quad (27) \\
& f(\pi(u_2) \mid (\sigma, d_1^j) \cdot u \mid \underline{v_1}) = f(\pi(u_2) \mid (\sigma, d_1^j) \cdot u \mid \underline{v_2}) \quad [(25), (26), (27)] \\
& \quad \quad \quad (28) \\
& f(\pi(u_2) \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_1}) = f(\pi(u_2) \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_2}) \quad [(24), (28), \text{Lemma 46}]
\end{aligned}$$

Since, $u_1 \equiv_f u_2$ and π satisfies all the conditions of Definition 7, we infer from the last equality above that $f(u_1 \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_1}) = f(u_1 \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_2})$. Hence, if $f(\pi' \odot \pi(u_2 \cdot (\sigma, d_2^j)) \cdot u \mid \underline{v_1}) = f(\pi' \odot \pi(u_2 \cdot (\sigma, d_2^j)) \cdot u \mid \underline{v_2})$, then $f(u_1 \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_1}) = f(u_1 \cdot (\sigma, d_1^j) \cdot u \mid \underline{v_2})$. Therefore, the permutation $\pi' \odot \pi$ satisfies all the conditions of Definition 7, so $u_1 \cdot (\sigma, d_1^j) \equiv_f u_2 \cdot (\sigma, d_2^j)$. ◀