Logic-Automata Connections for Transformations

Emmanuel Filiot*

Université Libre de Bruxelles

Abstract. Pioneered by Büchi, Elgot and Trakhtenbrot, connections between automata and logics that define languages of words and trees are now well-established. During the last decade, some of these powerful connections have been extended to binary relations (transformations) of words and trees. This paper is a survey of known automata-logic connections for transformations.

1 Introduction

The connections between mathematical logics and computational models have a long research history, which goes back to the foundations of theoretical computer science and the seminal works of Church and Turing [12,44]. In particular, Turing has shown how to express the behaviour of a universal machine in first-order logic, and then proved that first-order logic is undecidable, as a consequence of the undecidability of the halting problem. The Curry-Howard isomorphism is another important example of connection that shows correspondences between the formulas of a logic and the types of a computational model, and between proofs and programs [18,30].

Further connections between mathematical logic and automata theory have been discovered in the $60s$ by Büchi [10], Elgot [21] and Trakhtenbrot [43], who have shown that the class of finite word languages definable in monadic second-order logic corresponds, in an effective way, to the class of languages definable by finite state automata, and thus to regular languages. While logical formalisms have a high-level descriptive power, automata are easier to analyse algorithmically. For instance, checking whether the language defined by a finite state automaton is empty can be decided in linear-time. Therefore, as an application of Büchi-Elgot-Trakhtenbrot's theorem, monadic second-order logic (interpreted on finite words) has decidable satisfiability problem. Since this seminal result, many other similar connections have been shown, most notably for regular languages of infinite words and trees [11,36,37] and first-order definable languages of words [39]. More details can be found in the following survey: [42], [45] and [19].

A language of finite words over an alphabet Σ is a mapping from the set of words Σ^* to $\{0,1\}$. A *transformation* of finite words is a binary relation R on

* FNRS Research Associate (*Chercheur Qualifié*)

Σ^*, and therefore it generalises the concept of languages. It is *functional* if R is a function. Although transformations are as fundamental as languages, much less is known on the relation between automata and logic for transformations. Nevertheless, some important results have been obtained in the last decade. In this paper, we survey some of them.

A transformation R of finite words over an alphabet Σ can be seen as a language, for instance the language $\{u\#v \mid (u, v) \in R\}$, where $\# \notin \Sigma$. However, the formalisms from language theory, such as automata, are not well-suited to describe transformations defined on this encoding and therefore, proper extensions have been introduced to define transformations. Automata have been for instance extended to *automata with outputs*, usually called *transducers*. Perhaps the most studied transducer model is that of *finite state transducers* [32,38]. Finite state transducers (FST) extend finite state automata with an output mechanism. Whenever an FST reads an input symbol, it moves to the next symbol, updates its internal state, and write a partial output word. The final output word is the concatenation, taken in order, of all the partial output words produced while processing the whole input word.

The expressiveness of FST is limited and other, more powerful, state-based models have been introduced and studied, such as two-way transducers and more recently, streaming string transducers [2]. On the logic side, monadic-second order logic has been extended in a natural way to *MSO-transducers* by B. Courcelle, to define transformations of logical structures [14,16]. The predicates of the output structure are defined by MSO formulas interpreted over a fixed number of copies of the input structure. The first automata-logic connection, or one should say transducer-logic connection, has been shown in [22] by J. Engelfriet and H.J. Hoogeboom. They have extended Büchi-Elgot-Trakhtenbrot's theorem to functional transformations by showing that any transformation definable by a deterministic two-way finite state transducer is definable by an MSO-transducer (interpreted over finite words), and conversely. Moreover, this correspondence is effective, i.e., an MSO-transducer can be effectively constructed from a deterministic two-way finite state transducer and conversely. An important consequence of this result is the decidability of equivalence of MSO-transducers, since the equivalence problem for deterministic two-way transducers is decidable [17].

Since then, other transducer-logic connections have been established for finite word transformations and other structures such as infinite words and finite trees. Functional MSO-transformations of finite words have been shown to correspond to transformations definable by streaming string transducers [2], and this result has been extended to infinite words [3] and to non-functional MSO-transformations [22,5]. Engelfriet-Hoogeboom's theorem has been extended to finite trees [8,23,24]. First-order definable transformations of finite words have been considered in [27] and [33]. Some of these connections have been considered under a stronger semantics, the origin semantics, in [9].

This paper surveys some of these important results. All the transducer-logic connections presented in this paper are effective. The setting of functional transformations of finite words is presented in details, in contrast to the other results,

which come nevertheless with the main bibliographic references. In Section 2, we present some preliminary notions. In Section 3, we define first-order and monadic second-order logics interpreted on finite words, and define MSO-transducers, for which we give several examples. In Section 4, we introduce the main state-based models of transformations used in this paper. In Section 5, we present the main transducer-logic connection for transformations of finite words. Finally in Section 6, we briefly survey some extensions of the finite word setting.

2 Word Transformations

We define the preliminary notions used all over this paper.

Words An *alphabet* Σ is a finite set of symbols, called letters. A *word* w over Σ is a finite sequence of letters $(\sigma_1, \ldots, \sigma_n)$, denoted $w = \sigma_1 \ldots \sigma_n$. The empty word (empty sequence) is denoted by ϵ. The *length* of a non-empty word $w = \sigma_1 \ldots \sigma_n$ is defined by $|w| = n$, and $|\epsilon| = 0$. We denote by $\mathrm{dom}(w) = \{1, \ldots, |w|\} \subseteq \mathbb{N}^*$ the domain of w. In particular, $\mathrm{dom}(\epsilon) = \varnothing$. For all $i \in \mathrm{dom}(w)$, i is called a *position* of w and $w(i)$ denotes the i-th letter of w. The set of words over Σ is denoted by Σ^*, while the set of non-empty words over Σ is denoted by Σ^+.

Given two words $w_1 = \sigma_1 \ldots \sigma_n$ and $w_2 = \beta_1 \ldots \beta_m$, their *concatenation*, denoted $w_1.w_2$ (or simply $w_1 w_2$), is defined by $w_1.w_2 = \sigma_1 \ldots \sigma_n \beta_1 \ldots \beta_m$. In particular, $\epsilon w = w\epsilon = w$ for all words $w \in \Sigma^*$. For all $w \in \Sigma^*$ and $n \in \mathbb{N}$, we denote by w^n the concatenation of w, n times. In particular, $w^0 = \epsilon$, $w^1 = w$ and $w^2 = ww$.

Transformations A *transformation* R of finite words over an alphabet Σ is a binary relation over Σ^*, i.e. $R \subseteq \Sigma^* \times \Sigma^*$. For all words $u \in \Sigma^*$, we let $R(u)$ be the set of images of u by R, i.e. $R(u) = \{v \in \Sigma^* \mid (u, v) \in R\}$. The word u is usually called an *input word* while the words v such that $(u, v) \in R$ are called *output words*. We denote by $\mathrm{dom}(R)$ the domain of R, and by $\mathrm{range}(R)$ its range, i.e. $\mathrm{dom}(R) = \{u \in \Sigma^* \mid R(u) \neq \varnothing\}$ and $\mathrm{range}(R) = \{v \in \Sigma^* \mid \exists u \in \Sigma^*, v \in R(u)\}$.

A transformation R is *functional* if R is a function, i.e. for all words $u \in \Sigma^*$, the cardinality of $R(u)$ is smaller than or equal to 1, i.e. $|R(u)| \leq 1$. Functional transformations are rather denoted by $f, g, h \ldots$. For a functional transformation f, we write $f(u) = v$ instead of $f(u) = \{v\}$, for all $(u, v) \in f$.

Example 1. Let $\Sigma = \{a, b\}$. The following examples of (functional) transformations of finite words over Σ are running examples in this paper.

- The transformation $f_{del} : \Sigma^* \to \Sigma^*$ deletes all letters a, i.e. for all input words $u = \sigma_1 \ldots \sigma_n$, $f_{del}(u) = \sigma_{i_1} \ldots \sigma_{i_k}$ such that $\{i_1 < \cdots < i_k\} = \{i \in \mathrm{dom}(w) \mid w(i) \neq a\}$. E.g. $f_{del}(abaabb) = bbb$.
- The transformation f_{double} doubles every input letter, i.e. for all $u = \sigma_1 \ldots \sigma_n$, $f_{double}(u) = \sigma_1\sigma_1 \ldots \sigma_n\sigma_n$, e.g. $f_{double}(abaa) = aabbaaaa$.

- The transformation f_{copy} copies input words twice, i.e. for all $u \in \Sigma^*$, $f_{copy}(u) = uu$.
- The transformation f_{rev} reverses input words, i.e. $f_{rev}(\sigma_1 \dots \sigma_n) = \sigma_n \dots \sigma_1$. E.g. $f_{rev}(abaa) = aaba$.
- The transformation $f_{1/2}$ is defined over a^* by, for all $n \geq 0$, $f_{1/2}(a^n) = a^{\lfloor n/2 \rfloor}$. E.g. $f_{1/2}(a^8) = a^4$ and $f_{1/2}(a^3) = a$.
- The transformation f_{exp} exponentiates the number of a symbols in a word of the form a^n, e.g. $f_{exp}(a^n) = a^{2^n}$, and $f_{exp}(w)$ is undefined if w contains at least one b.

3 Logical Transducers for Word Transformations

In this section, we introduce *logical transducers*, a logic-based formalism introduced by B. Courcelle [15] to define transformations of logical structures. We refer the reader to [16] for more details and results about logical transducers. Although logical transducers can generally define transformations of arbitrary logical structures, we specialise them to finite word transformations in this section. We first introduce the notion of word logical structures, and then the classical first-order and monadic second-order logics, interpreted over (logical structures of) words.

3.1 Words as logical structures

A word w over an alphabet Σ can be seen as a *logical structure*[1] $\tilde{w}$ over the signature $\mathcal{S}_\Sigma = \{(L_a)_{a\in\Sigma}, \preceq\}$, where $(L_a)_{a\in\Sigma}$ are monadic predicates that define the labels of the positions in w, and $\preceq$ is a binary predicate that defines the order on word positions. Formally, $\tilde{w} = (\mathrm{dom}(w), (L_a^{\tilde{w}})_{a\in\Sigma}, \preceq^{\tilde{w}})$ is the logical structure whose domain is $\mathrm{dom}(w)$, and such that the predicates are interpreted as follows:

$$L_a^{\tilde{w}} = \{i \in \mathrm{dom}(w) \mid w(i) = a\} \qquad \preceq^{\tilde{w}} = \{(i,j) \mid i,j \in \mathrm{dom}(w) \wedge i \leq j\}$$

When it is clear from the context, we rather write w instead of $\tilde{w}$.

A structure on $\mathcal{S}_\Sigma$ is also called a $\mathcal{S}_\Sigma$-structure. We denote by $\mathcal{M}(\mathcal{S}_\Sigma)$ the set of $\mathcal{S}_\Sigma$-structures. Note that a $\mathcal{S}_\Sigma$-structure may not be isomorphic to any word. However for all $w \in \Sigma^*$, $\tilde{w} \in \mathcal{M}(\mathcal{S}_\Sigma)$. Given a structure in $M \in \mathcal{M}(\mathcal{S}_\Sigma)$, we denote by $\mathrm{dom}(M)$ its domain.

3.2 First-order and monadic second-order logics on words

Given an alphabet Σ, *monadic second-order formulas* (*MSO formulas*) over the signature $\mathcal{S}_\Sigma$ are built over first-order variables $x, y \dots$ and second-order variables $X, Y \dots$. They are defined by the following grammar:

$$\phi \ ::= \ \exists X \cdot \phi \mid \exists x \cdot \phi \mid \phi \wedge \phi \mid \neg\phi \mid x \in X \mid L_a(x) \mid x \preceq y \mid (\phi)$$

[1] See for instance [20] or [41] for a definition of logical structures.

Universal quantifiers and other Boolean connectives are defined naturally: $\forall x \cdot \phi \equiv \neg\exists x \cdot \neg\phi$, $\forall X \cdot \phi \equiv \neg\exists X \cdot \neg\phi$, $\phi_1 \vee \phi_2 \equiv \neg(\phi_1 \wedge \phi_2)$ and $\phi_1 \to \phi_2 \equiv \neg\phi_1 \vee \phi_2$. We also define the formulas true and false: $\top \equiv \forall x \cdot (L_a(x) \vee \neg L_a(x))$ and $\bot \equiv \neg\top$. We do not define the semantics of MSO formulas, neither the standard notion of free and bound variables, but rather give examples and refer the reader to [20] or [41] for formal definitions.

Given an MSO formula ϕ, we write $\phi(x_1, \ldots, x_n, X_1, \ldots, X_m)$ to emphasise the fact that the free first-order variables of ϕ are exactly $x_1, \ldots, x_n$, and its free second-order variables are $X_1, \ldots, X_m$. Given a $\mathcal{S}_\Sigma$-structure M and an MSO sentence ϕ, we write $M \models \phi$ when M satisfies ϕ. Let $i_1, \ldots, i_n \in \mathrm{dom}(M)$, $I_1, \ldots, I_m \subseteq \mathrm{dom}(M)$. For a formula $\phi(x_1, \ldots, x_n, X_1, \ldots, X_m)$, we write $M \models \phi(i_1, \ldots, i_n, I_1, \ldots, I_m)$ to denote the fact that M together with the interpretation of x_j by i_j, $j = 1, \ldots, n$ and X_j by I_j, $j = 1, \ldots, m$, satisfy ϕ.

Given an MSO sentence ϕ, we write $[\![\phi]\!]$ the set of words that satisfy ϕ, i.e. $[\![\phi]\!] = \{w \in \Sigma^* \mid \tilde{w} \models \phi\}$. Given a language $L \subseteq \Sigma^*$, if there exists an MSO sentence ϕ such that $[\![\phi]\!] = L$, we say that L is MSO-definable, and that ϕ *defines* L.

First-order logic First-order (FO) formulas over $\mathcal{S}_\Sigma$ are MSO formulas in which no second-order variable occurs.

Example 2. Let $\Sigma = \{a, b\}$. The formula $\exists x \cdot \top$ defines the set of non-empty words. The formula $\exists x \cdot L_a(x)$ define the set of words over Σ that contain at least one position labelled a, i.e. the language $\Sigma^* a \Sigma^*$.

The formula $S(x, y) \equiv x \preceq y \wedge x \neq y \wedge \forall z \cdot (x \preceq z \preceq y \to (x = z \vee y = z)$ defines the successor relation.

The formula $\forall x \forall y \cdot (L_a(x) \wedge S(x, y) \to L_b(y))$ defines the set of words such that any occurrence of the letter a is followed by the letter b. The formulas

$$\mathrm{first}(x) \equiv \neg\exists y \cdot S(y, x) \qquad \text{and} \qquad \mathrm{last}(x) \equiv \neg\exists y \cdot S(x, y)$$

are such that for all $w \in \Sigma^+$ and $i \in \mathrm{dom}(w)$, $w \models \mathrm{first}(i)$ iff $i = 1$, and $w \models \mathrm{last}(i)$ iff $i = |w|$.

The language $a^* b^*$ is definable by the following formula:

$$\forall x \forall y \cdot (L_a(x) \wedge S(y, x) \to L_a(y))$$

More generally, it is known that the class of MSO-definable languages is the class of regular languages [42]. The MSO formula

$$\mathrm{part}(X_1, \ldots, X_n) \equiv (\forall x \cdot \bigvee_{i=1}^{n} x \in X_i) \wedge \forall x \cdot \bigwedge_{i \neq j} (x \notin X_i \vee x \notin X_j)$$

holds true whenever $X_1, \ldots, X_n$ defines a partition of the domain. Finally, one can define the set of words of even length in MSO, but one needs second-order

variables X_o and X_e to capture, respectively, odd and even positions of the word, as defined by the formula

$$\begin{aligned}\phi_{o/e}(X_o, X_e) \equiv \mathrm{part}(X_o, X_e) \wedge \forall x \cdot (\mathrm{first}(x) \to x \in X_o) \\ \wedge \forall x \forall y \cdot S(x,y) \to (x \in X_o \to y \in X_e) \wedge (x \in X_e \to y \in X_o)\end{aligned}$$

Then, the set of words of even length is defined by the sentence

$$\exists X_o \exists X_e \cdot \phi_{o/e}(X_o, X_e) \wedge \forall x \cdot (\mathrm{last}(x) \to x \in X_e).$$

3.3 Logical transducers: definition

Logical transducers define functional transformations from input to output word structures. The output structure is defined by taking a fixed number k of copies of the input structure domain. Some node of these copies can be filtered out by formulas with one free first-order variable. In particular, the nodes of the c-th copy are the input positions that satisfy some given formula $\phi^c_{pos}(x)$. The predicates L_a and $\preceq$ of the output structure are defined by formulas with respectively one and two free first-order variables, interpreted over the input structure. More precisely, position labelled a of the c-th copy are defined by a given formula $\phi^c_{L_a}(x)$, interpreted over the input word. If this formula holds true, it means that the c-th copy of x, if it exists, is labelled a in the output word. The order relation between two output positions is defined by formulas with two free variables interpreted over the input word. For instance, the order relation between positions of the c-th copy and the d-th copy (c and d can be equal) is defined by a formula $\phi_{\preceq}(x,y)^{c,d}$ interpreted over the input structure. If this formula holds true, it means the the c-th copy of x occurs before the d-th copy of y in the output word. Let us formally define logical transducers.

Definition 1. *Let Σ be an alphabet. A logical MSO-transducer (MSOT) on the signature $\mathcal{S}_\Sigma$ is a tuple*

$$T = (k, \phi_{dom}, (\phi^c_{pos}(x))_{1 \le c \le k}, (\phi^c_{L_a}(x))_{1 \le c \le k, a \in \Sigma}, (\phi^{c,d}_{\preceq}(x,y))_{1 \le c,d \le k})$$

where $k \in \mathbb{N}$ and the formulas ϕ_{dom}, ϕ^c_{pos}, $\phi^c_{L_a}$ and $\phi^{c,d}_{\preceq}$ for all $c, d \in \{1, \ldots, k\}$ and $a \in \Sigma$ are MSO formulas over $\mathcal{S}_\Sigma$.

Semantics A logical MSO-transducer T defines a function from $\mathcal{S}_\Sigma$-structures to $\mathcal{S}_\Sigma$-structures, denoted by $[\![T]\!] : \mathcal{M}(S_\Sigma) \to \mathcal{M}(S_\Sigma)$. The domain of $[\![T]\!]$ consists of all structures M such that $M \models \phi_{dom}$. Given a structure $M \in dom([\![T]\!])$, the *output structure* N such that $(M,N) \in [\![T]\!]$ is defined by $N = (D^N, (L^N_a)_{a \in \Sigma}, \preceq^N)$ where:

- $D^N \subseteq \mathrm{dom}(M) \times \{1, \ldots, k\}$ is defined by

$$D^N = \{(i,c) \mid i \in \mathrm{dom}(M),\ c \in \{1, \ldots, k\},\ M \models \phi^c_{pos}(i)\}$$

  We rather denote by i^c the elements of D^M.

– for all $a \in \Sigma$, the interpretation L_a^N is defined by

$$L_a^N = \{i^c \mid i \in \mathrm{dom}(M),\ c \in \{1,\dots,k\},\ M \models \phi^c_{L_a}(i)\} \cap D^N$$

– the interpretation $\preceq^N$ is defined by

$$\preceq^N = \{(i^c, j^d) \mid i,j \in \mathrm{dom}(M),\ c,d \in \{1,\dots,k\},\ M \models \phi^{c,d}_{\preceq}(i,j)\} \cap (D^N \times D^N)$$

Remark 1. Note that the size of the output structure N is linearly bounded by the size of M, as it is at most $k.|dom(M)|$. We say that MSO-transducers define *linear-size increase* transformations.

Logical transducers as word-to-word transformers Note that in general, an MSO-transducer T over $\mathcal{S}_\Sigma$ may not define a word-to-word transformation, as the output structure of an input word structure may not be a word. We say that T is an *MSO-transducer of finite words over* Σ if for all words $w \in \Sigma^*$ such that $\tilde{w} \in \mathrm{dom}(T)$, $[\![T]\!](\tilde{w})$ is a word, i.e., there exists $v \in \Sigma^*$ such that $[\![T]\!](\tilde{w})$ is isomorphic to $\tilde{v}$. This property is decidable:

Proposition 1. *It is decidable whether an MSO-transducer over $\mathcal{S}_\Sigma$ is an MSO-transducer of finite words over Σ.*

Proof. Let $T = (k, \phi_{dom}, (\phi^c_{pos}(x))_{1\le c\le k}, (\phi^c_{L_a}(x))_{1\le c\le k, a\in\Sigma}, (\phi^{c,d}_{\preceq}(x,y))_{1\le c,d\le k})$. We construct a formula $\mathrm{is\_word}_T$ which is satisfiable in Σ^* iff T is an MSO-transducer of finite words over Σ. The result follows since MSO over finite words is decidable, by Büchi-Elgot-Trakhtenbrot's Theorem.

Before giving the construction, let us introduce the following useful shortcuts. We write $\forall x^c \cdot \phi$ instead of $\forall x \cdot \bigwedge_{c=1}^k \phi$ and $\exists x^c \cdot \phi$ instead of $\exists x \cdot \bigvee_{c=1}^k \phi$. We also write $[\forall x^c] \cdot \phi$ instead of $\forall x^c \cdot (\phi^c_{pos}(x) \to \phi)$ to mean that x^c is quantified over output nodes that belong to the domain of the output structure. By $x^c = y^d$ we denote the formula $x = y$ if $c = d$, and $\bot$ if $c \neq d$. Therefore, by $x^c \neq y^d$ we denote the formula $x \neq y$ if $c = d$, and $\top$ if $c \neq d$.

It is also convenient to define the output successor relation. For all $c, d \in \{1,\dots,k\}$, we let

$$\phi^{c,d}_S(x,y) \equiv \phi^{c,d}_{\preceq}(x,y) \wedge x^c \neq y^d \wedge \forall z^e \cdot (\phi^{c,e}_{\preceq}(x,z) \wedge \phi^{e,d}_{\preceq}(z,y) \to z^e = x^c \vee z^e = y^d)$$

Finally, we can construct the expected formula:

$$\begin{aligned}
\mathrm{is_word}_T \equiv\ & \phi_{dom} \to \\
& (1) \quad [\forall x^c] \cdot \textstyle\bigwedge_{a\neq b\in\Sigma} \neg\phi^i_{L_a}(x) \vee \neg\phi^i_{L_b}(x) \\
\wedge\ & (2) \quad [\forall x^c] \cdot \textstyle\bigvee_{a\in\Sigma} \phi^c_{L_a}(x) \\
\wedge\ & (3) \quad [\forall x^c \forall y^d \forall z^e] \cdot (\phi^{c,d}_S(x,y) \wedge \phi^{c,e}_S(x,z) \to y^d = z^e) \\
\wedge\ & (4) \quad [\forall x^c \forall y^d \forall z^e] \cdot (\phi^{d,c}_S(y,x) \wedge \phi^{e,c}_S(z,x) \to y^d = z^e) \\
\wedge\ & (5) \quad [\forall x^c \forall y^d] \cdot (x^c \neq y^d) \to ([\exists z^e] \cdot \phi^{e,c}_S(z,x) \vee [\exists z^e] \cdot \phi^{e,d}_S(z,y)) \\
\wedge\ & (6) \quad [\exists x^c \forall y^d] \cdot \neg\phi^{d,c}_S(y,x)
\end{aligned}$$

Subformula (1) ensures that each output node is labeled by at most one letter. Subformula (2) ensures that each output node is labeled by at least one letter. Subformula (3) ensures that the output successor relation is a function. Subformula (4) ensures that the inverse of the output successor relation is a function. Finally, Subformulas (5) and (6) ensures that there is exactly one output node without predecessor. □

In the rest of this section, by MSO-transducer and MSOT we always mean an MSO-transducer of finite words.

FO-transducers of finite words An FO-transducer (FOT) T is defined as an MSO-transducer, except that each formula of T is an FO formula over $\mathcal{S}_\Sigma$.

Definability We say that a transformation R of finite words is definable by a logical transducer T if $R = [\![T]\!]$. We say that R is MSOT-definable (resp. FOT-definable) if it is definable by an MSO-transducer (resp. FO-transducer) of finite words.

3.4 Logical transducers: Examples

In this section, we give several examples of transformations that can be defined by MSO-transducers.

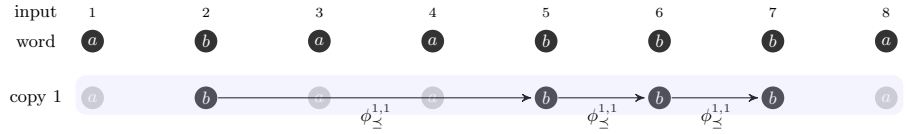
Example 3. We show that all transformations of Example 1 but f_{exp} are MSOT-definable. They are illustrated in Fig. 1. Only the successor relations are depicted. Input nodes filtered out by formulas $\phi^c_{pos}(x)$ are represented by fuzzy nodes.
• The transformation f_{del} on $\Sigma = \{a, b\}$ is definable by the transducer

$$T_{del}{=}(1, \phi_{dom} \equiv \top, \phi^1_{pos}(x) \equiv \neg L_a(x), (\phi^1_{L_\sigma}(x) \equiv L_\sigma(x))_{\sigma\in\Sigma}, \phi^{1,1}_{\preceq}(x, y) \equiv x \preceq y)$$

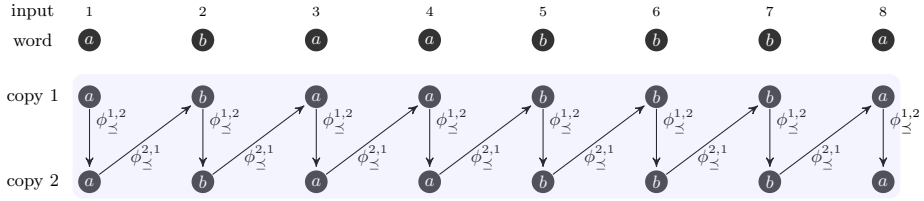
Given an input word $u \in \Sigma^*$, let $v \in \Sigma^*$ such that $\tilde{v} = [\![T_{del}]\!](\tilde{u})$. Then $dom(\tilde{v}) = \{i^1 \in \mathrm{dom}(u) \mid u(i) = b\}$, as defined by ϕ^1_{pos}, and $\preceq^{\tilde{v}} = \{(i^1, j^1) \mid i^1, j^1 \in \mathrm{dom}(\tilde{v}), i \leq j\}$.
• To define transformation f_{double}, one needs to take two copies of the input structure. It is defined by the transducer T_{double} with $k = 2$ and for $i \in \{1, 2\}$:

$$\begin{array}{cccc} \phi_{dom} \equiv \top & \phi^i_{pos}(x) \equiv \top & \phi^i_{L_a}(x) = L_a(x) & \phi^i_{L_b}(x) = L_b(x) \\ \phi^{1,1}_{\preceq}(x, y) \equiv x \preceq y & \phi^{1,2}_{\preceq}(x, y) \equiv x \preceq y & \phi^{2,1}_{\preceq}(x, y) \equiv x \prec y & \phi^{2,2}_{\preceq}(x, y) \equiv x \preceq y \end{array}$$

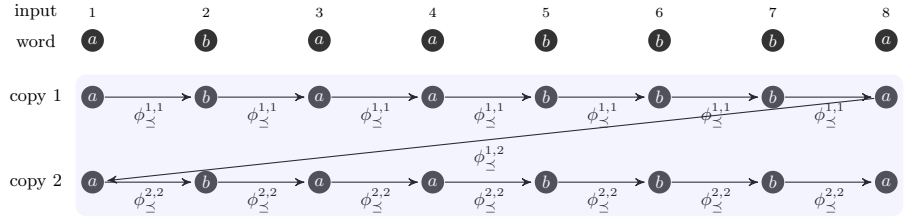
Note that the output predicate $\preceq$ from copy 2 to copy 1 is only defined when x occurs strictly before y. It implies that an output node y^d is a successor of x^d iff one of the two following conditions hold: (i) $c = 1$ and $d = 2$ and $x = y$, or (ii) $c = 2$ and $d = 1$ and y is a successor of x in the input word. If one wants to restrict the domain of T_{double} to words in a^*, it suffices to define the domain formula by $\phi_{dom} \equiv \forall x \cdot L_a(x)$.



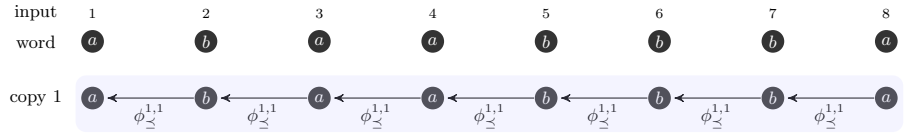
(a) Transformation f_{del} defined by T_{del}



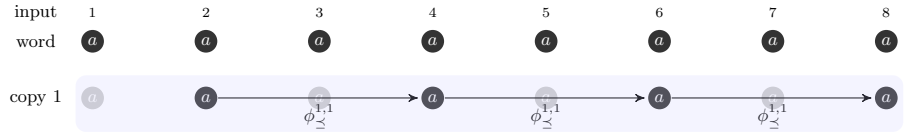
(b) Transformation f_{double} defined by T_{double}



(c) Transformation f_{copy} defined by T_{copy}



(d) Transformation f_{rev} defined by T_{rev}



(e) Transformation $f_{1/2}$ defined by $T_{1/2}$

Fig. 1. Transformations of Example 1 defined by MSO-transducers

• Let us consider transformation f_{copy}. Again, one needs two copies of the input structure. It is similar to T_{double} except the way the output order is defined:

$$\begin{array}{cccc} \phi_{dom} \equiv \top & \phi^i_{pos}(x) \equiv \top & \phi^i_{L_a}(x) = L_a(x) & \phi^i_{L_b}(x) = L_b(x) \\ \phi^{1,1}_{\preceq}(x,y) \equiv x \preceq y & \phi^{1,2}_{\preceq}(x,y) \equiv \top & \phi^{2,1}_{\preceq}(x,y) \equiv \bot & \phi^{2,2}_{\preceq}(x,y) \equiv x \preceq y \end{array}$$

Note that compared to T_{double}, only the definition of $\phi^{2,1}_{\preceq}$ differs. We indeed completely disallow a node from copy 2 to be smaller than a node from copy 1.
• The transformation f_{rev} is defined by the transducer T_{rev}: it suffices to take only one copy of the input structure and to inverse the order relation. Formally, T_{rev} is defined by $k = 1$ and:

$$\phi_{dom} \equiv \top \quad \phi^i_{pos}(x) \equiv \top \quad \phi^1_{L_a}(x) = L_a(x) \quad \phi^1_{L_b}(x) = L_b(x) \quad \phi^{1,1}_{\preceq}(x, y) \equiv y \preceq x$$

• To define with an MSO-transducer the transformation $f_{1/2} : a^n \mapsto a^{\lfloor n/2 \rfloor}$, one takes one copy of the input domain, sets the domain formula to $\phi_{dom} \equiv \forall x \cdot L_a(x)$, and filters out all odd positions of the input word, which is possible by the MSO formula $\phi^1_{pos}(x) \equiv \exists X_o \exists X_e \cdot \phi_{o/e}(X_o, X_e) \wedge x \in X_e$, where $\phi_{o/e}$ has been defined in Example 2. Finally, the order relation is just defined by $\phi^{1,1}_{\preceq}(x, y) \equiv x \preceq y$.
• The transformation $f_{exp} : a^n \mapsto a^{2^n}$ is not MSOT-definable, because it is not linear-size increase, while MSOT-definable transformations are, by Remark 1.

Remark 2. Let us mention an other, logic-based, transformation formalism, called *first-order translations*, that has been introduced by N. Immerman in [31], as a way to define reductions between problems. In first-order translations, the domain of the output structure is a set of k-tuples of elements of the input domain, for some k. It is defined by a first-order formula with k free variables. Predicates of arity n of the output structure are defined, similarly to Courcelle's logical transducers, by formulas with kn free variables, interpreted over the input structure. In contrast to logical transducers which are linear-size increase, first-order translations can map a structure to a polynomially larger output structure. First-order translations have been introduced as a logical way to define reductions between decision problems and nothing is known about their expressiveness as a formalism to define transformations. In this paper, we rather focus on (Courcelle) logical transducers, for which connections with state-based formalisms have been established. Nevertheless, let us mention the two papers [33] and [35], where the particular case of length-preserving FO-translations with $k = 1$ has been studied, as well as their connections with finite state transducers. See Section 5.2 for more details.

4 State-based Models for Word Transformations

In this section, we introduce some of the main state-based models for defining (finite) word transformations for which connections with logics are known. These models are automata models extended with outputs, and are usually called *transducers*. We present three models: finite state transducers, two-way finite state transducers, and streaming string transducers.

4.1 Finite state transducers

Finite state transducers (FST) extend finite state automata with partial output words on their transitions. Whenever an FST reads an input letter, it moves

(a) f_{del} (b) f_{double} (c) $f_{1/2}$

Fig. 2. Examples of finite state transducers.

deterministically to the next state and appends a word to the output tape. Formally, an FST on an alphabet Σ is a tuple $T = (Q, q_0, F, \delta)$ such that Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is a set of accepting states, and $\delta : Q \times \Sigma \to Q \times \Sigma^*$ is the transition function.

A *run* of T is a sequence $r = p_0\sigma_1 p_1 \dots \sigma_n p_n \in (Q\Sigma)^*Q$ such that $p_0 = q_0$ and for all $i \in \{1, \dots, n\}$, there exists $v_i \in \Sigma^*$ such that $\delta(p_{i-1}, \sigma_i) = (p_i, v_i)$. Given $u \in \Sigma^*$, one says that r is a *run on* u if $u = \sigma_1 \dots \sigma_n$. The output of r, denoted by $O(r)$, is defined as the word $O(r) = v_1 \dots v_n$. The run r is *accepting* if $p_n \in F$.

An FST T realises a functional transformation $[\![T]\!] : \Sigma^* \to \Sigma^*$ defined by

$$[\![T]\!] \;=\; \{(u, v) \mid \text{there exists an accepting run } r \text{ of } T \text{ on } u \text{ such that } v = O(r)\}$$

Note that indeed, since T is deterministic, $[\![T]\!]$ is a function. The extension of FST with non-determinism allows one to define relations instead of functions.

A *non-deterministic finite state transducer* (NFT) over an alphabet Σ is a tuple $T = (Q, q_0, F, \Delta)$ where Q, q_0, F are defined as for FST, and $\Delta : Q \times \Sigma \times Q \to \Sigma^*$ is a (partial) function that defines the transitions[2]. A run of T is defined similarly as a run of an FST, as a sequence $r = p_0\sigma_1 p_1 \dots \sigma_n p_n$ such that $p_0 = q_0$ and for all $i \in \{1, \dots, n\}$, $\delta(p_{i-1}, \sigma_i, p_i)$ exists and is equal to some $v_i \in \Sigma^*$. The output $O(r)$ is defined by $O(r) = v_1 \dots v_n$. The other notions defined for FST carry over to NFT. Note that $[\![T]\!]$ may not be a function, since there can be several accepting runs on an input word. However, whether an NFT defines a function is decidable in PTime (see, for instance, [7]). NFT defining functions are known as *functional NFT*.

Example 4. Fig. 2 illustrates three FST that define the functions f_{del}, f_{double} and $f_{1/2}$ respectively. On these figures, the vertical arrow represents the initial state, the double circles the accepting states, and the arrows labelled $\sigma \mid v$ the transitions that read $\sigma \in \Sigma$ and produce $v \in \Sigma^*$. The other functions, f_{copy}, f_{rev} and f_{exp} are not definable by finite state transducers (even NFT). As we will see in Section 5.3, any NFT-definable functional transformation is definable by an MSO-transducer. We define in Section 5.3 a restriction on MSO-transducer that captures exactly NFT-definable functions.

[2] These NFT are sometimes called *real-time* NFT, in contrast to a more general class of NFT that allow productive ϵ-transitions.

4.2 Two-way finite state transducers

Two-way finite state transducers (2FST) extend (one-way) finite state transducer with a bidirectional input head. Depending on the current state and letter, a 2FST updates its internal state and moves its input head either left or right. In order to detect the first and last positions of the input word, 2FST are assumed to run on words that are nested with begin and end markers $\vdash, \dashv$ respectively.

Formally, a *two-way finite state transducer (2FST)* over an alphabet Σ is a tuple $T = (Q, q_0, \delta, \delta_{halt})$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, and δ is the transition relation[3], of type $\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to Q \times \{+1, -1\} \times \Sigma^*$, such that $\delta(q, \vdash) \in Q \times \{+1\} \times \Sigma^*$, and $\delta(q, \dashv) \in Q \times \{-1\} \times \Sigma^*$, for all $q \in Q$. Finally, δ_{halt} is the halting function, of type $\delta_{halt} : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to \Sigma^*$. In order to ensure determinism, it is required that $\mathrm{dom}(\delta) \cap \mathrm{dom}(\delta_{halt}) = \varnothing$.

In order to see how a word $u \in \Sigma^*$ is evaluated by T, it is convenient to see the input as a tape containing $\vdash u \dashv$. Initially the head of T is on the first cell in state q_0 (the cell labelled $\vdash$). When T reads an input symbol, depending on the transitions in δ, its head moves to the left (-1) if the head was not in the first cell, or to the right ($+1$) if the head was not in the last cell, then it updates its state, and appends a partial output word to the final output. T stops as soon as it can apply the halting transition δ_{halt}, and produces a last partial output word.

A *configuration* of T is a pair $(q, i) \in Q \times \mathbb{N}$ where q is a state and i is a position on the input tape. A *run* r of T is a finite sequence of configurations. Let $u = \sigma_1 \dots \sigma_n \in \Sigma^*$, let $\sigma_0 = \vdash$ and let $\sigma_{n+1} = \dashv$. A run $r = (p_1, i_0) \dots (p_m, i_m)$ is *accepting on* u if (i) $p_1 = q_0$, $i_0 = 0$; (ii) $\delta_{halt}(p_m, \sigma_{i_m})$ is defined and equal to v_m for some $v_m \in \Sigma^*$; (iii) for all $k \in \{0, \dots, m-1\}$, $\delta(p_k, \sigma_{i_k})$ is defined and equal to $(p_{k+1}, i_{k+1} - i_k, v_k)$ for some $v_k \in \Sigma^*$. The *output* of r is defined by $O(r) = v_1 \dots v_m$. Like FST, the (functional) transformation defined by T, denoted by $[\![T]\!]$, is the set of pairs (u, v) such that there exists an accepting run r of T on u such that $O(r) = v$.

Example 5. Unlike FST, 2FST can define the functions f_{copy} and f_{rev}, as shown in Fig. 3. Therefore, there are strictly more expressive than FST. However, checking whether a 2FST is equivalent to some FST is decidable [26]. 2FST define linear-size increase transformations, because it can be proved that due to determinism, the number of times an input position can be visited is in $O(|Q|)$. Therefore, f_{exp} is not 2FST-definable.

4.3 Streaming string transducers

[3] In the literature, some definitions also include stay transition, i.e. transitions where the input head does not move. In the deterministic case, these transitions can however be removed without loss of expressiveness.

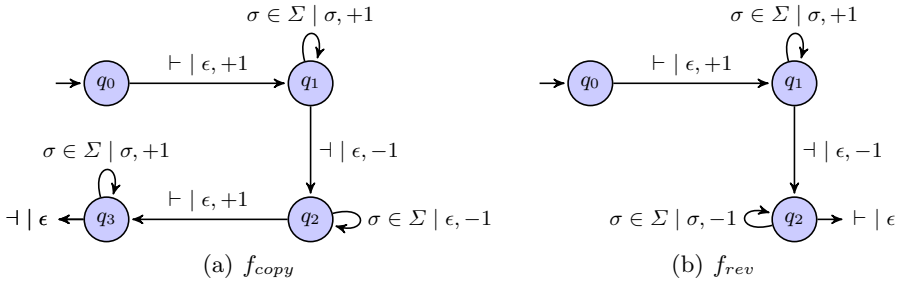


Fig. 3. Examples of two-way finite state transducers.

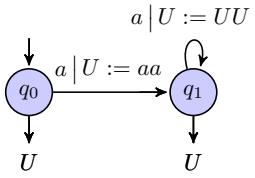


Fig. 4. SST for f_{exp}

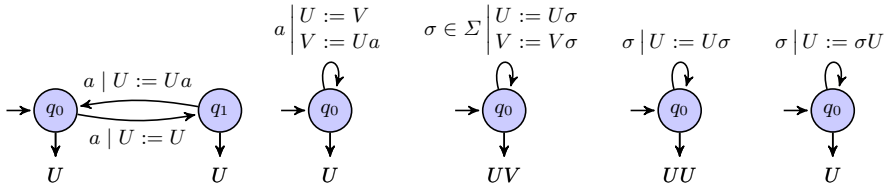
Recently, an appealing transducer model for word transformations, whose expressiveness is exactly the same as 2FST, has been proposed in [2], as an extension of FST with registers, called *streaming string transducers (SST)*. Partial output words are stored in a fixed number of registers that can be concurrently updated and combined in different ways to define the output word. Moreover, SST are deterministic and, unlike 2FST, are one-way (left-to-right), making them easier to manipulate algorithmically. It is has been applied, for instance, to the automatic verification of some important classes of list-processing programs [1].

Let $\mathcal{X}$ be a finite set of registers denoted by the capital letters $U, V, W \ldots$ and Σ be an alphabet. A substitution s is defined as a mapping $s : \mathcal{X} \to (\Sigma \cup \mathcal{X})^*$. A valuation is defined as a substitution $v : \mathcal{X} \to \Sigma^*$. Let $\mathcal{S}_{\mathcal{X},\Sigma}$ be the set of all substitutions. Any substitution s can be extended to $\hat{s} : (\Sigma \cup \mathcal{X})^* \to (\Sigma \cup \mathcal{X})^*$ in a straightforward manner. The composition $s_1 s_2$ of two substitutions s_1 and s_2 is defined as the standard function composition $\hat{s_1} \circ s_2$.

A *streaming string transducer* (SST) over Σ is a tuple $T = (Q, q_0, \delta, \mathcal{X}, \rho, O)$ where Q is a finite set of states with initial state q_0; $\delta : Q \times \Sigma \to Q$ is a transition function; $\mathcal{X}$ is a finite set of registers; $\rho : \delta \to \mathcal{S}_{\mathcal{X},\Sigma}$ is a register update function; and $O : Q \rightharpoonup \mathcal{X}^*$ is a (partial) output function.

Like FST, a *run* r of an SST T is an alternating sequence of states and letters $r = p_0\sigma_1 p_1 \ldots \sigma_n p_n$ such that $p_0 = q_0$ and for all $i \in \{0, \ldots, n-1\}$, $\delta(p_i, \sigma_{i+1})$ is defined and equal to p_{i+1}. The run r is *accepting* if $p_n \in \mathrm{dom}(O)$. We let $|r| = n$ the length of r. In particular, a run of length 1 follows exactly one transition. The sequence $\langle s_{r,i} \rangle_{0 \leq i \leq |r|}$ of substitutions induced by r is defined inductively as: $s_{r,0}$ is the identity function over $\mathcal{X}$, and $s_{r,i} = s_{r,i-1}\rho(p_{i-1}, \sigma_i)$ for $1 \leq i \leq |r|$. We denote $s_{r,|r|}$ by s_r.

If the run r is accepting, we can extend the output function O to the run r by $O(r) = s_\epsilon s_r(O(p_n))$, where s_ϵ substitutes all registers by their initial value ϵ.



(a) $f_{1/2}$ (b) $f_{1/2}$ (c) f_{copy} (d) f_{copy} (e) f_{rev}

Fig. 5. Examples of streaming string transducers

As for FST and 2FST, the functional transformation $[\![T]\!]$ defined by an SST T is the set of pairs (u, v) such that there exists an accepting run r of T on u such that $v = O(r)$.

Example 6. The definition of SST is best understood with some examples. Any FST can be encoded as an SST with a single register. As an example, consider the SST that defines the function $f_{1/2}$ in Fig. 5(a). It has only one register U. The register update substitutions are represented on the edges by the the assignment operator $:=$, while the output function is represented by the vertical arrows leading to an expression, here U. The function $f_{1/2}$ can also be defined with only one state, but using one additional register V, as depicted by Fig. 5(b). The function f_{copy} can be defined with one or two registers, as depicted by Fig. 5(c) and Fig. 5(d). Finally, f_{rev} is defined by the SST of Fig. 5(e). Unlike MSO-transformations, SST-definable transformations may not be linear-size increase, as shown by the SST of Fig.4 which defines the transformation f_{exp}. To capture MSO-transformations, various syntactic restrictions on SST register updates have been defined in several papers [2,3,4,6], which can be defined as restrictions of a uniform notion of *transition monoids* for SSTs [27], as presented in the next section.

4.4 Transition monoids for streaming string transducers

The transition monoid of an SST is a set of matrices M_w, for all words $w \in \Sigma^*$, that represent the state and variable flow of the SST over w [27]. Let $T = (Q, q_0, \delta, \mathcal{X}, \rho, O)$ be an SST over an alphabet Σ, let two states $q_1, q_2 \in Q$, two registers $U_1, U_2 \in \mathcal{X}$, a word $w \in \Sigma^*$ and $n \in \mathbb{N} \cup \{\bot\}$. Intuitively, if $n \geq 0$, the pair (q_1, U_1) n-flows to (q_2, U_2) on reading w if there exists a run of T on w from q_1 to q_2, on which the sequence of register updates makes U_1 contributes n times to the content of U_2. The pair (q_1, U_1) $\bot$-flow to (q_2, U_2) if there is no run on w from q_1 to q_2. For example, for the SST of Fig. 5(a), (q_0, U) 1-flows to (q_1, U) on reading a, as well as a^k for all odd integers k. On Fig. 5(b), (q_0, U) 1-flows to (q_0, V) on reading a. On Fig. 4, (q_0, U) 2^k-flows to (q_0, U) on reading a^k for all $k \geq 0$.

Formally, (q_1, U_1) *n-flows to* (q_2, U_2) on w $(n \geq 0)$, denoted $(q_1, U_1) \overset{w}{\underset{n}{\rightsquigarrow}} (q_2, U_2)$, if there exists a run r of T on w, from state q_1 to state q_2, such that

U_1 occurs n times in $s_r(U_2)$, where s_r is the substitution defined by r. The pair (q_1, U_2) $\perp$-flows to (q_2, U_2) if there is no run of T on w, from state q_1 to state q_2. We denote by M_w the $(\mathbb{N} \cup \{\perp\})$-valued square matrices of dimension $|Q|.|\mathcal{X}|$ defined, for all $q_1, q_2 \in Q$ and all $U_1, U_2 \in \mathcal{X}$, and all $n \in \mathbb{N} \cup \{\perp\}$, by

$$M_w[q_1, U_1][q_2, U_2] = n \text{ iff } (q_1, U_1) \overset{w}{\underset{n}{\rightsquigarrow}} (q_2, U_2)$$

Definition 2 (SST transition monoid). *The transition monoid of an SST T is the set of matrices $\mathcal{M}(T) = \{M_w \mid w \in \Sigma^*\}$.*

Note that $\mathcal{M}(T)$ is indeed a monoid with matrix multiplication and M_ϵ as neutral element (M_ϵ is equal to the identity matrix on $Q \times \mathcal{X}$). Moreover, the mapping $w \in \Sigma^* \mapsto M_w$ is a morphism, as it can be shown that $M_{w_1 w_2} = M_{w_1} M_{w_2}$ for all $w_1, w_2 \in \Sigma^*$ [27].

Classes of transition monoids We define several classes of transition monoids. The transition monoid $\mathcal{M}(T)$ of an SST T is *copyless* if for all $M \in \mathcal{M}(T)$, M is $\{\perp, 0, 1\}$-valued, and every row $M[q, U][.]$ contains at most one 1, for all $q \in Q$ and $U \in \mathcal{X}$. In other words, U can be copied in at most one other register (included itself). The monoid $\mathcal{M}(T)$ is *restricted copy* if all $M \in \mathcal{M}(T)$ is $\{\perp, 0, 1\}$-valued. In other words, a register can be copied more than once, but these copies must not be combined later on. Finally, we will also consider SST whose transition monoid is *finite*. The registers with a finite transition monoid can be copied, but not on loops. Note that any copyless transition monoid is restricted copy, and any restricted copy transition monoid is finite. It has been shown, as we will see, that the corresponding classes of SST are, however, of equal expressive power.

Several restrictions on register updates that have been defined in several papers, with ad-hoc definitions, are nicely captured by these simple classes of transition monoids. The copyless restriction of [2] corresponds to SST with copyless transition monoid. The restricted copy restriction of [6] corresponds to SST with restricted copy transition monoids. Finally, the bounded copy restriction of [3] corresponds to SST with finite transition monoid.

5 Automata-logic connections for word transformations

In this section, we present the main known automata-logic connections for word transformations.

5.1 MSO transformations

The first automata-logic connection for word transformations has been discovered by J. Engelfriet and H.J. Hoogeboom [22]:

Theorem 1 (J. Engelfriet, H.J. Hoogeboom [22]). *Let $f : \Sigma^* \to \Sigma^*$. The function f is MSOT-definable iff it is 2FST-definable.*

The proof of MSOT⇒2FST is based on intermediate models of 2FST which can perform "MSO jumps" $\phi(x, y)$, where $\phi(x, y)$ is an MSO formula that defines a function from x positions to y positions. Intuitively, the machine can move from position x to position y providing $\phi(x, y)$ holds true. 2FST with MSO jumps are then converted, based on Büchi-Elgot-Trakhtenbrot's theorem, into 2FST with regular look-around. These 2FST can move only to positions which are in the 1-neighbourhood of the current position, but their move can be based on a regular property of the current prefix and suffix of the word. Finally, it is shown that 2FST with regular look-around are equivalent to 2FST.

The converse, 2FST⇒MSOT, is shown by first constructing an MSOT that takes as input a word structure, and output an edge-labeled graph (edges are labelled by words of bounded length) whose nodes are exactly the configurations (q, i) in which the 2FST is successively (where q is a state and i a position in the input word). This MSOT is then composed with an MSOT that transforms an edge-labeled graph into a node-labeled graph (nodes are here labelled with letters from Σ). The result follows as MSOT are closed under composition [16,15].

As we have seen in Fig. 4, SST can define functions which are exponential-size increase (f_{exp}), and therefore, not MSOT-definable. However, any MSOT-definable transformation is SST-definable, and, by weakening the expressiveness of SST, it is possible to capture exactly MSOT:

Theorem 2 (R. Alur, P. Černý [2], R. Alur, E. Filiot, A. Trivedi [3]). *Let $f : \Sigma^* \to \Sigma^*$. The following statements are equivalent:*

1. *f is MSOT-definable.*
2. *f is definable by an SST with copyless transition monoid.*
3. *f is definable by an SST with finite transition monoid.*

The equivalence between (1) and (2) was shown in [2]. The proof of (1) ⇒ (2) of [2] goes through the intermediate model of 2FST and relies on Theorem 1. More precisely, it is shown that any 2FST can be encoded as an SST, by extending to transducers the classical Sheperdson's construction that transforms a two-way finite automaton as a (one-way) finite automaton [40]. The resulting SST may not be necessarily copyless, and the main challenge is to show that it can be converted into a copyless SST. Conversely, it is shown how to directly encode an SST as an MSO-transducer. In [3], it has been shown that SST with finite transition monoid (called bounded copy) are equivalent to SST with copyless transition monoid. Although the proof of (1) ⇒ (2) in [2] relies on 2FST, a direct construction was also given in [6], in which the states of the SST are MSO-types of bounded quantifier rank.

5.2 FO transformations

Recall that first-order transformations are transformations definable by FO-transducers, which are defined as MSO-transducers, except that only first-order formulas can be used. In automata theory for languages, first-order definable

languages are captured by *aperiodic automata*, i.e. finite automata whose transition monoid is aperiodic [41,19]. A survey on first-order definable languages can be found in [19]. A monoid $(M, \cdot, e)$ (with operator $\cdot$ and neutral element e) is *aperiodic* if there exists $k \in \mathbb{N}$ such that for all $m \in M$, $m^k = m^{k+1}$. Recall that the language $(aa)^*$ is not FO-definable, while the language $(ab)^*$ is.

As we have seen, the functions f_{del}, f_{double}, f_{copy}, and f_{rev} are FOT-definable. However, the function $f_{1/2}$ is not FOT-definable [27], although its domain, a^*, is. So clearly, the FOT-definability of a function not only depends on the FO-definability of its domain. It can be seen on an example. In Fig. 5(a), the transition monoid of the underlying automaton of the SST defining $f_{1/2}$ (the automaton obtained by dropping the register updates) is not aperiodic. Therefore, in order to get FOT-definable functions, a first restriction would be to require that the underyling automaton of an SST is aperiodic. However, it is not sufficient, as shown by the SST of Fig. 5(b), whose underlying automaton has aperiodic transition monoid. However, the register flow, which alternates between U and V registers on reading a, is not aperiodic. If one requires that the transition monoid of an SST, which also speaks about the register flow, is aperiodic, then one gets exactly FOT-definable functions:

Theorem 3 (E. Filiot, K. Shankara Narayanan and A. Trivedi [27]). *Let $f : \Sigma^* \to \Sigma^*$. The function f is FOT-definable iff it is definable by an SST with aperiodic and restricted copy transition monoid.*

This theorem should not be understood as an effective characterization of FOT-definable functions. Indeed, it could be the case that an SST which defines an FOT-definable function has not an aperiodic transition monoid. As an example, consider an SST which alternates on reading the letter a between two states, both accepting, and realizes the identity function with a single register. Its transition monoid is not aperiodic, but the function it defines if FOT-definable.

It is also the case for automata: a non-aperiodic automaton may define a first-order language. However for automata, FO-definable languages can be algebraically characterised, as show by M.P. Schützenberger : a language L is FO-definable iff its syntactic monoid is aperiodic [39]. This characterisation is effective if L is given as a finite automaton, and decidable in PSPace [19].

Finally, like for automata, deciding whether the transition monoid of an SST is aperiodic and restricted copy is PSPace-C [27].

FO-translations in one-free variable Let us mention another transducer-logic connection that has been shown for a less expressive class of functions, namely the FO-translations of [31] restricted to FO-formulas in one-free variable and length-preserving functions. Such a translation assumes a total order $<$ on Σ and is defined by a tuple T of FO-formulas in one-free variable x, say $T = (\phi_\sigma(x))_{\sigma \in \Sigma}$, such that $\phi_{\sigma_m}(x) = \bigwedge_{\sigma < \sigma_m} \neg \phi_\sigma(x)$, where σ_m is the maximal element of Σ. Then, for all $w \in \Sigma^*$ of length n, $T(w) = \sigma_1 \dots \sigma_n$ such that for all $i \in \{1, \dots, n\}$, $w \models \phi_{\sigma_i}(i) \wedge \bigwedge_{\sigma < \sigma_i} \neg \phi_\sigma(i)$. Note that T defines a total and length-preserving function, which is actually definable by a one-copy (Courcelle) FO-transducer.

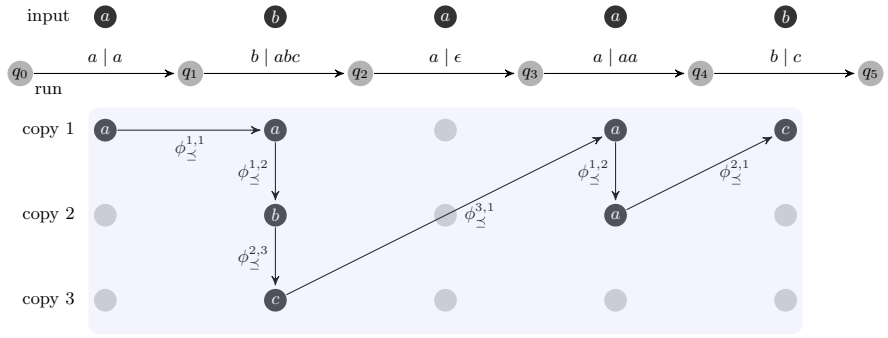


Fig. 6. Encoding of NFT by MSO-transducer

It is shown in [33] that a function is definable by an FO-translation in one-free variable iff it is definable by an aperiodic NFT, where aperiodic NFT are the NFT whose underlying automata is aperiodic (i.e. they have aperiodic transition monoid). This result was later on, in [35], generalised to V-translations and NFT whose transition monoid is in V, where V is a pseudovariety of (finite) monoids (for instance, the pseudovariety of finite aperiodic monoids).

5.3 Order-preserving MSO transducers

An MSO-transducer T with k copies is *order-preserving* if for all words $w \in \mathrm{dom}(T)$, all positions $i, j \in \mathrm{dom}(w)$, and copies $c, d \in \{1, \ldots, k\}$, if $w \models \phi_{\preceq}^{c,d}(i,j)$, then $i \preceq j$ holds true. Note that this can be syntactically ensured by requiring that formulas $\phi_{\preceq}^{c,d}(x,y)$ are of the form $x \preceq y \wedge \phi$. We show in Theorem 4 that order-preserving MSO-transducers characterise exactly the NFT-definable functions. This theorem can also be obtained as a consequence of a result shown in [9] for order-preserving transformations with origin semantics. However, we give here a direct proof, which illustrates the techniques that are usually used to derive automata-logic connections. Origin semantics is discussed later in Section 5.4.

Theorem 4. *A function $f : \Sigma^* \to \Sigma^*$ is NFT-definable iff it is definable by an order-preserving MSO-transducer.*

Proof. We first show the "only if" direction. Let $T = (Q, q_0, F, \Delta)$ be an NFT over Σ that defines a f. We construct an order-preserving MSO-transducer T' such that $[\![T]\!] = [\![T']\!] = f$. Since T defines a function, it is known that T is equivalent to an unambiguous NFT [32], i.e. an NFT such that there exists at most one accepting run on every input word. Therefore, we assume that T is unambiguous.

Let $K \in \mathbb{N}$ be the maximal length of the words occurring on the transitions of T, i.e. $K = \max\{|w| \mid \exists p, q \in Q \exists \sigma \in \Sigma,\ \Delta(p, \sigma, q) = w\}$. Clearly, for all words $u \in \mathrm{dom}(T)$, $|[\![T]\!](u)| \leq K.|u|$. In order to define $[\![T]\!]$ by an MSO-transducer,

one needs to take K copies of the input structure. Let us intuitively explain how these copies will be used. Let $u = \sigma_1 \dots \sigma_n \in \mathrm{dom}(T)$ and $r = q_0\sigma_1 q_1 \dots \sigma_n q_n$ be **the** accepting run of T on u, and let $O(r) = v_1 \dots v_n$, where each word v_i is produced by the i-th transition of r. Every position $j \in \mathrm{dom}(v_i)$ will be encoded by the j-th copy of the input position i, i.e., by the node i^j. Of course, the j-th copy exists since $|v_i| \leq K$. For all j such that $|v_i| < j \leq K$, one needs to filter out all nodes i^j, by the formula $\phi^j_{pos}(x)$. The encoding is illustrated on Fig. 6.

The definition of T' relies on the existence of MSO formulas $\phi_t(x)$, for all tuples $t = (p, \sigma, v, q) \in Q \times \Sigma \times \Sigma^* \times Q$, such that for all words $u \in \Sigma^*$ and all positions $i \in \mathrm{dom}(u)$, $u \models \phi_t(i)$ iff $u \in \mathrm{dom}(T)$ and the run of T on u, say $r = q_0\sigma_1 q_1 \dots \sigma_n q_n$, is such that $q_{i-1} = p$, $\sigma_i = \sigma$, $q_i = q$ and $\Delta(p, \sigma, q) = v$. We do not prove the existence of such formulas in this paper, it is obtained as a direct consequence of Büchi-Elgot-Trakhtenbrot's Theorem. Another direct consequence of this theorem is the existence of an MSO formula $\phi_{\mathrm{dom}(T)}$ which defines the domain of T (which is a regular language).

We define formally $T' = (k, \phi_{dom}, (\phi^c_{pos})_{1 \leq c \leq k}, (\phi^c_{L_\sigma})_{1 \leq c \leq k, \sigma \in \Sigma}, (\phi^{c,d}_{\preceq})_{1 \leq c,d \leq k})$:

$$
\begin{array}{ll}
k & = K \\
\phi_{dom} & \equiv \phi_{\mathrm{dom}(T)} \\
\phi^c_{pos}(x) & \equiv \bigwedge\{\phi_t(x) \to |v| \geq c \mid t := (p, \sigma, v, q) \text{ s.t. } \Delta(p, \sigma, q) = v\} \\
\phi^c_{L_\sigma}(x) & \equiv \bigwedge\{\phi_t(x) \to (c \leq |v| \wedge v(c) = \sigma) \mid t := (p, \sigma, v, q) \text{ s.t. } \Delta(p, \sigma, q) = v\} \\
\phi^{c,d}_{\preceq}(x, y) & \equiv (x = y \wedge c \leq d) \vee x \prec y
\end{array}
$$

where the subformulas $|v| \geq c$, $c \leq |v| \wedge v(c) = \sigma$ and $c \leq d$ (which do not actually belong to MSO syntax) are either defined by $\top$ or $\bot$ depending on whether the Boolean expressions they represent hold true or not. Finally, note that T' is indeed order-preserving since clearly, if $u \models \phi^{c,d}_{\preceq}(i, j)$, then $i \leq j$.

Conversely, given an order-preserving MSO-transducer T, one shows how to construct an equivalent NFT T'. We start by a first observation. Let $u = \sigma_1 \dots \sigma_n \in \mathrm{dom}(T)$ and $v = [\![T]\!](u)$. Since T is order-preserving, there is no backward edges in the output word structures produced by T. Therefore the word v can be decomposed into $v_1 \dots v_n$ such that each word v_i is the subword of v induced by the non-filtered copies of the input position i, i.e. the set $N_i = \{i^1, \dots, i^k\} \cap \{i^j \mid u \models \phi^j_{pos}(i)\}$. It can be seen on Fig. 6: each vertical block of the output structure correspond to a partial output word, and the result output word of the transformation is obtained by concatenating, in order, these partial words. Let Σ^*_k be all the words over Σ of length at most k. For all $w \in \Sigma^*_k$, one can define an MSO formula $\Psi_w(x)$ such that $u \models \Psi_w(i)$ iff $u \in \mathrm{dom}(T)$ implies that $w = v_i$, where v_i is defined by the decomposition explained before. Note that the definition of order-preservation does not require that copies are ordered according to $\leq$ (on integers), i.e., whenever formula $\phi^{c,d}_{\preceq}(x, y)$ holds, it implies that $x \preceq y$, but not necessarily $c \leq d$, unlike the example of Fig. 6. In other words, it could be that the first position of v_i corresponds to node i^2 while the second position corresponds to node i^1 of the output structure. Therefore, in order to define $\Psi_w(x)$, one has to quantify, by a huge disjunction, over possible orders over these copies. Formally, for all tuples $t = (j_1, \dots, j_{|w|})$ such that $j_\ell \in \{1, \dots, k\}$

for all $\ell \in \{1, \ldots, |w|\}$, and $j_{\ell_1} \neq j_{\ell_2}$ for all $\ell_1 \neq \ell_2 \in \{1, \ldots, |w|\}$, we define an intermediate formula $\Psi_w^{(j_1,\ldots,j_{|w|})}(x)$ which holds true whenever the x-th partial output word is w, and each position of w correspond to nodes x^{j_ℓ}. Then, $\Psi_w(x)$ is obtained as the disjunction of the formulas $\Psi_w^{(j_1,\ldots,j_{|w|})}(x)$ for all tuples:

$$\Psi_w^t(x) \equiv \bigwedge_{\ell=1}^{|w|} \phi_{pos}^{j_\ell}(x) \wedge \phi_{L_{w(\ell)}}^{j_\ell}(x) \wedge \bigwedge_{j \in \{1,\ldots,k\} \setminus \{j_1,\ldots,j_{|w|}\}} \neg \phi_{pos}^{j}(x) \bigwedge_{\ell=1}^{|w|-1} \phi_{\preceq}^{j_\ell, j_{\ell+1}}(x, x)$$
$$\Psi_w(x) \equiv \bigvee \{\Psi_w^t(x) \mid t = (j_1, \ldots, j_{|w|}) \in \{1, \ldots, k\}^{|w|}, j_{\ell_1} \neq j_{\ell_2},\ \forall \ell_1 \neq \ell_2\}$$

In other words, for all $u \in \mathrm{dom}(T)$, we have $u \models \Psi_w(i)$ iff w is the i-the subword of the output of u. This formula is of particular interest if we want to define an NFT, because it tells us exactly what partial output word must be produced while reading the i-th input letter.

The rest of the proof explains how one can construct an automaton A on the alphabet $\Gamma = \Sigma \times \Sigma_k^*$ which accepts all words $(\sigma_1, v_1) \ldots (\sigma_n, v_n) \in \Gamma^*$ such that for all $i \in \{1, \ldots, n\}$, v_i is the (unique) word such that $\sigma_1 \ldots \sigma_n \models \Psi_{v_i}(i)$. The final NFT T' is obtained from this automaton A by replacing every transition $(q, (\sigma, v), p)$ of A by the transition $q \xrightarrow{\sigma|v} p$ of T'. Clearly, T' is equivalent to T.

In order to construct A, we again apply Büchi-Elgot-Trakhtenbrot's theorem, since the language of A is MSO-definable, by the sentence (on $\mathcal{S}_\Gamma$)

$$\phi_A \equiv [\phi_{dom}]_\Gamma \wedge \forall x \cdot \bigwedge_{(\sigma,v) \in \Gamma} L_{(\sigma,v)}(x) \rightarrow [\Psi_v(x)]_\Gamma$$

where $[\phi_{dom}]_\Gamma$ (resp. $[\Psi_v(x)]_\Gamma$) is obtained from ϕ_{dom} (resp. $\Psi_v(x)$), which is on the signature $\mathcal{S}_\Sigma$, by replacing every atom $L_\beta(y)$ by $\bigvee_{w \in \Sigma_k^*} L_{(\beta,w)}(y)$. Clearly, a word $s = (\sigma_1, v_1) \ldots (\sigma_n, v_n)$ over $\Sigma \times \Sigma_k^*$ satisfies $[\phi_{dom}]_\Gamma$ iff $u = \sigma_1 \ldots \sigma_n \models \phi_{dom}$. Similarly, $s \models [\Psi_v(i)]_\Gamma$ iff $u \models \Psi_v(i)$, and therefore the correctness follows.□

Again, the power of transducer-logic connections is illustrated by the following definability problem. It is decidable whether a deterministic two-way finite state transducer T is equivalent to a (one-way) functional finite state transducer [26]. As a consequence of this result, the connection between 2FST and MSOT (Theorem 1), and the previous theorem (Theorem 4), we obtain the following corollary:

Corollary 1. *Given an MSO-transducer T, it is decidable whether T is equivalent to some order-preserving MSO-transducer.*

Proof. It suffices to translate T into a deterministic two-way transducer T', by applying the (effective) encoding of Theorem 1, then to apply the procedure of [26] which decides whether T' is equivalent some NFT, and finally to apply Theorem 4. □

5.4 Transformations with origin

One of the difficulties in the theory of transducers is to deal with the asynchronicity in the production of the output. For instance, two FST may not produce there output letters at the same time when reading the same input position, but still, they can be equivalent. This asynchronicity, which is implicitly present in the Post correspondence problem (PCP), can even, in some cases, lead to undecidable problems. For instance, the equivalence problem of non-deterministic FST is undecidable, and the undecidability proof is non surprisingly based on PCP [29]. The asynchronicity between output words have been captured by a notion of *output delay*, which have been used, for instance, to make elegant proofs of the decidability of equivalence of functional FST [7].

Recently introduced by M. Bojanczyk, a stronger semantics have been given to transformations, that takes into account the origin of the output letters in the input word. Several classical problems have been revisited and most of them become trivial with this new semantics [9]. Some problems, still open in the classical semantics, have also been solved in the origin semantics, such as getting a machine-independent characterisation of MSO-transformations, as well as an effective characterisation of first-order transformations. We present some of these new results in this section.

Definition 3. *Let Σ be an alphabet, and $u, v \in \Sigma^*$. An origin function for v in u is a mapping $o : \mathrm{dom}(v) \to \mathrm{dom}(u)$. A word transformation with origin over Σ is a set of pairs $(u, (v, o))$ such that $u, v \in \Sigma^*$ and o is an origin function for v in u.*

Intuitively, $o(i)$ is the position in the input word at which the position i of the output word has been produced. Origin semantics can be defined for the transducer models we have seen so far. For instance, the origin transformation (or o-transformation) defined by an FST T is the set of pairs, denoted by $[\![T]\!]_o$, of the form $(u, (v, o))$ such that $(u, v) \in [\![T]\!]$, and for all $i \in \mathrm{dom}(v)$, $o(i)$ the position of u where T has produced $v(i)$, i.e. where T has triggered a transition on reading $u(o(i))$ which has produced a partial word containing the letter $v(i)$.

With this semantics, the origin equivalence problem for FST, i.e. deciding whether two FST T_1, T_2 satisfy $[\![T_1]\!]_o = [\![T_2]\!]_o$, can be easily solved, because T_1 and T_2 can be seen respectively as two automata A_1, A_2 over $\Sigma \times \Sigma_k^*$, where k is the longest output word occurring on the transitions of T_1 and T_2. Indeed, $[\![T_1]\!]_o = [\![T_2]\!]_o$ iff the automata A_1 and A_2 are equivalent, i.e., define the same language. This trick, however, cannot be used for two-way FST, making origin semantics more interesting in this context.

It is also possible to give a natural origin semantics to transformations realised by MSO-transducers T. In that case, $(u, (v, o)) \in [\![T]\!]_o$ if $(u, v) \in [\![T]\!]_o$ and, if i^c is the k-th node of v (wrt to $\preceq$), where $i \in \mathrm{dom}(u)$ and c is a copy, then we let $o(k) = i$.

In the origin world, it is possible to characterise algebraically first-order definable transformations with origin, while this problem is open in the classical

setting. Moreover, this characterisation is effective when the transformation is defined by, say, an MSO-transducer.

Let $u \in \Sigma^*$, $o : \mathrm{dom}(u) \to \mathbb{N}$, and $X \subseteq \mathbb{N}$. The abstraction $u/_X$ of u by X is the word over $\Sigma_\perp := \Sigma \cup \{\perp\}$ obtained by replacing in u each letter at position $i \in \mathrm{dom}(u)$ by $\perp$ if $o(i) \in X$, where $\perp$ is a fresh symbol not in Σ. For instance, if $u = aba$ with origin $o(1) = 3$, $o(2) = 2$ and $o(3) = 1$, then $u/_{\{2,3\}} = \perp\perp a$.

We let $\sim_\perp$ the equivalence relation on $\Sigma_\perp^*$ induced by the equation $\perp = \perp\perp$. E.g. $\perp a \perp b \sim_\perp \perp\perp a \perp\perp\perp b$, but $a \not\sim_\perp \perp a$.

Given an o-transformation f, one defines its reverse $rev(f)$. For all $u \in \Sigma^*$, if $(v, o) = f(u)$, then $rev(f) = (f_{rev}(v), rev(o))$, where $rev(o)(i) = o(|v| - i + 1)$.

Similarly to the syntactic equivalence relation for languages, one can define *left-* and *right-* equivalence relations for transformations with origin f, resp. denoted by $\mathcal{L}^f$ and $\mathcal{R}^f$.

Definition 4. *Let f be a transformation with origin on Σ. Let $v_1, v_2 \in \Sigma^*$. Then, $v_1 \mathcal{L}^f v_2$ if for all $w \in \Sigma^*$:*

1. *$v_1 w \in \mathrm{dom}(f)$ iff $v_2 w \in \mathrm{dom}(f)$*
2. *if $v_1 w \in \mathrm{dom}(f)$, then $f(v_1 w)/_{\mathrm{dom}(v_1)} \sim_\perp f(v_2 w)/_{\mathrm{dom}(v_2)}$.*

Symmetrically, $\mathcal{R}^f = \mathcal{L}^{rev(f)}$.

Example 7. For instance, consider the o-transformation g that maps any $u \in \Sigma^*$ to $(f_{rev}(u)u, o)$, where, naturally, for all $i \in \{1, \ldots, |u|\}$, $o(i)=|u|-i+1$ and $o(|u|+i)=i$. Then, there are only two equivalence classes for both equivalence relation: $\{\epsilon\}$ and Σ^+. Indeed, for all $w \in \Sigma^*$,

$$f(v_1 w)/_{\mathrm{dom}(v_1)} = (f_{rev}(w) f_{rev}(v_1) v_1 w)/_{\mathrm{dom}(v_1)} = f_{rev}(w) \perp^{2|v_1|} w.$$

Therefore $f(v_1 w)/_{\mathrm{dom}(v_1)} \sim_\perp f_{rev}(w)w$ if $v_1 = \epsilon$ and $f(v_1 w)/_{\mathrm{dom}(v_1)} \sim_\perp f_{rev}(w) \perp w$ otherwise. A similar arguments applies for $\mathcal{R}^f$.

It is then possible to characterise MSOT-definable o-transformations by a Myhill-Nerode like theorem, and to give an effective characterisation of FOT-definable o-transformations.

Theorem 5 (M. Bojanczyk [9][4]). *Let f be a transformation with origin over Σ. The following two statements hold true:*

1. *f is MSOT-definable iff both $\mathcal{L}^f$ and $\mathcal{R}^f$ have finite index.*
2. *f is FOT-definable iff both $\mathcal{L}^f$ and $\mathcal{R}^f$ have finite index and for all $u \in \Sigma^*$, the $\mathcal{L}^f$-class of u and the $\mathcal{R}^f$-class of u are FO-definable languages.*

As shown in [9], if f is MSOT-definable, then the $\mathcal{L}^f$- and $\mathcal{R}^f$-equivalence classes are regular languages that can be effectively represented by automata. Characterization (2) is therefore effective, as it suffices to decide whether all equivalence classes are FO-definable, which is decidable [19].

[4] We adopt in this paper a slightly different, but equivalent, formalisation as in [9].

6 Beyond functional finite word transformations

In this section, we discuss other transducer-logic connections for non-functional transformations and other structures (infinite words and trees).

6.1 Non-functional finite word transformations

The state-based transducer models (FST, 2FST and SST) can all be extended with non-determinism (leading to the classes NFT, 2NFT, and NSST resp.), and rather define relations from word to words instead of functions. MSO-transducers can, as well, be extended with non-determinism.

Non-deterministic state-based models We have already seen in Section 4.1 how to extend FST with non-determinism. Similarly, 2FST can also be extended with non-determinism, the transition relation Δ being of type $\Delta \subseteq Q \times (\Sigma \cup \{\dashv, \vdash\}) \times Q \times \{+1, -1\} \times \Sigma^*$. Unlike FST, there can be infinitely many runs on the same input word, leading to infinitely many outputs for that word. For instance, 2NFT can loop an arbitrary number of times between two input positions and non-deterministically decide to apply the halting transition function. Their runs can even be infinite but we consider only finite runs to define the transformation.

SST can, as well, be extended with non-determinism, with transition relation $\Delta \subseteq Q \times \Sigma \times Q$. Unlike 2NFT, there is alway a finite number (exponential in the worst-case) of accepting runs on the same input word.

Non-deterministic MSO-transducers (NMSOT) MSO-transducers can be extended with non-determinism by allowing all the formulas (including the domain formula) to use a finite set of second-order of variables $X_1, \ldots, X_n$. Given an input word u, the outputs of u depend on the valuations of these second order variables by subsets of $\mathrm{dom}(u)$. In particular, modulo a valuation of $X_1, \ldots, X_n$ by subsets of $\mathrm{dom}(u)$, the transformation becomes functional, and the outputs of u are the set of output words defined for each valuation. For instance, it is possible to define with an NMSO-transducer T_{sub} the tranformation R_{sub} which maps any word u to all its subwords, using one second-order variable X and only one copy, as follows: $\phi_{dom} \equiv \top$, $\phi^1_{pos}(x, X) = x \in X$, $\phi^1_\sigma(x) = L_\sigma(x)$ for all $\sigma \in \Sigma$, and $\phi^{1,1}_{\preceq}(x, y) \equiv x \preceq y$. If one wants to restrict all the subwords to subwords generated by the even positions, it suffices to strengthen the domain formula to $\phi_{dom}(X) \equiv \exists Y \phi_{o/e}(Y, X)$, where $\phi_{o/e}$ has been defined in Example 2.

Transducer-Logic Connections A transformation R is *finitary* if for all words $u \in \Sigma^*$, $R(u)$ is finite. It is clear that NFT, NSST and NMSOT define finitary transformations. However, 2NFT does not define, in general, finitary transformations. It turns out that 2NFT and NMSOT define incomparable classes of transformations.To capture exactly NMSOT with a two-way device, *Hennie machines* have been introduced in [22]. Hennie machine can rewrite their input tape, but each input position must be visited a constant number of times. On the one-way model side, it has been shown that NSTT corresponds exactly to NMSOT [5].

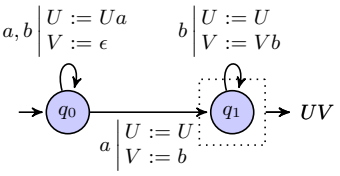


Fig. 7. Example of ω-SST defining f_ω, with output function $O(\{q_1\}) = UV$.

6.2 Infinite word transformations

An *ω-word* over Σ is a mapping $w : \mathbb{N} \to \Sigma$. The i-th letter of w is $w(i)$. An infinite words w over Σ is either a finite word or an ω-word. The set of ω-words over Σ is denoted by Σ^ω, and the set of infinite words by Σ^∞. Note that $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. A transformation R of infinite words over Σ is a binary relation from Σ^ω to Σ^∞ (we assume indeed in this section that the input word is an ω-word).

MSO-transducers can naturally be generalised to define functional infinite word transformations, by seeing an ω-word as a structure over $\mathcal{S}_\Sigma$ whose domain is $\mathbb{N}$. For an MSO-transducer T to be an MSO-transducer of infinite words, we require that for all input words $u \in \Sigma^\omega$, the image $[\![T]\!](u)$ is a structure that corresponds to an infinite word. Although it is a semantical restriction, it is decidable, similarly as in the proof of Proposition 1.

As an example, consider the transformation $f_\omega : \Sigma^\omega \to \Sigma^\omega$, where $\Sigma = \{a, b\}$, that maps any input word of the form uab^ω to $a^{|u|}b^\omega$, where $u \in \Sigma^*$. It is definable by the following MSO-transducer with one copy and $\phi_{dom} \equiv \exists x \cdot L_a(x)$,

$$\phi^1_{pos}(x) \equiv \top \quad \phi^1_a(x) \equiv \exists y \succ x \cdot L_a(y) \quad \phi^1_b(x) \equiv \neg\phi^1_a(x) \quad \phi^{1,1}_{\preceq}(x, y) \equiv x \preceq y$$

(Deterministic) 2FST can be extended to define functional infinite word transformations, by using for instance a Muller acceptance condition (they are called ω-2FST). However, they cannot even define f_ω, because they can never decide locally whether a b letter should be transformed into an a letter or kept unchanged, because it depends on the existence of an a letter in the future. They could use the two-wayness as a kind of look-ahead to check the existence of such an a, but they cannot come back exactly to the position they were coming from, because they get lost, due to the finite state device. One therefore needs to extend ω-2FST with regular look-ahead: each transition of an ω-2FST with regular look-ahead is extended with a finite state automaton over ω-words that checks a property of the (infinite) suffix. Such transition can be triggered only if the suffix belong to the look-ahead automaton. It should be clear that ω-2FST with regular look-ahead strictly extend the expressive power of ω-2FST.

SST have been extended to define functional infinite word transformations, with a Muller accepting condition (called ω-SST). They run on ω-words in a deterministic way, and the (partial) output function O has type $2^Q \to \mathcal{X}^*$. Given a run r over an ω-word u, let P the set of states visited infinitely many

times in r. The output of r is defined only if $O(P)$ is defined, as the limit of the sequence of finite words $s_\epsilon s_{r,i}(O(P))$ for $i \to \infty$ (remind that s_ϵ and $s_{r,i}$ have been defined in Section 4.3). In order to ensure the existence of that limit (and to make sure that this limit is an infinite word), syntactic restrictions are put on the SST: if $P \in \mathrm{dom}(O)$ and $O(P) = U_1 \dots U_n$, it is required that on the connected component induced by P in T, the registers $U_1, \dots, U_{n-1}$ are never modified, and the register U_n can only be modified by appending something (i.e. updates are of the form $U_n := U_n\alpha$ for some $\alpha \in (\Sigma \cup \mathcal{X})^*$). The transition monoid of an ω-SST is defined similarly as SST.

As an example, consider the ω-SST of Fig. 7 that defines transformation f_ω. It can loop an arbitrary number of steps in state q_0 while replacing all symbols by a, and storing the current output in register U. Non-deterministically, it guesses the last occurrence of an input a symbol, and from that point on, never modifies register U again, and always append b to register V. Register U is intended to capture the word $a^{|u|}$ in the definition of f_ω, while V captures b^ω. The output is then UV, and it is defined only for the singleton $\{q_1\}$, which enforces that after some time, only b symbols are read.

It has been shown in [3] that MSO-transducers of infinite words correspond exactly to ω-2FST with regular look-ahead, to ω-SST with finite transition monoid, and to ω-SST with copyless transition monoid.

6.3 Tree transformations

We present a result that establishes a correspondence between MSO-transducers, and a transducer model for functional transformations of finite ranked trees. Recall that ranked trees are ordered trees over a ranked alphabet. Each symbol f of the alphabet has a rank denoted by $r(f)$ and, if some node α is labelled f, this node has exactly $r(f)$ successor nodes, called the children of α (see [13] for a formal definition). We denote by $\mathcal{T}_\Gamma$ the set of ranked trees over a ranked alphabet Γ, and a (ranked) tree transformation is a binary relation over $\mathcal{T}_\Gamma$.

Tree transducers and their connection with term rewriting systems have been deeply studied, see for instance [28]. More recent results on tree transducers can be found in [16]. Like words, ranked trees over Γ can be seen as logical structures over the signature $\mathcal{S} = \{S_1, \dots, S_n, (L_a)_{a\in\Gamma}\}$, where n is the maximum arity in Γ, S_i are binary successor predicates interpreted by pairs of nodes (α, β) such that β is the i-th child of α, and L_a are unary predicates for the node labels. An MSO-transducer over the signature $\mathcal{S}$ defines a functional tree transformation on $\mathcal{T}_\Gamma$, provided the output is a ranked tree structure (which is a decidable property). For instance, consider a ranked alphabet $\Gamma = \{g, a\}$ where g is a binary symbol and a a constant, and the transformation t_{rev} which reverses a tree, i.e., reverses the order relation between the children of any internal node. The transformation t_{rev} is definable by the following one-copy MSO-transducer:

$$\phi_{dom} \equiv \phi^1_{pos}(x) \equiv \top \quad \phi^1_{L_\gamma}(x) \equiv L_\gamma(x) \quad \phi^{1,1}_{S_i}(x,y) \equiv \bigvee_{\gamma\in\Gamma} L_\gamma(x) \wedge \phi_{S_{r(\gamma)-i+1}}(x,y)$$

for $\gamma \in \Gamma$ and $i \in \{1, 2\}$.

Correspondences between MSO-transducers on ranked trees and tree transducers has been first studied in [23,8,24] for attribute grammars and macro tree transducers, and more recently in [4] for streaming tree transducers.

Macro tree transducers extend top-down tree transducers with parameters in which to store partial output trees. They correspond to purely functional programs working on tree structures: states are mutually recursive functions and can carry parameters. Due to lack of space, we do not define formally macro tree transducers. MSO-transducers on ranked trees correspond exactly to (deterministic) macro tree transducers of linear-size increase, i.e. macro tree transducers that define functions whose output tree size depends linearly on the size of the input tree. It is shown in [23,8] that functional MSOT-transformations of ranked trees are definable by macro tree transducers. The other, more difficult direction, which shows that macro tree transducers of linear-size increase are effectively MSOT-definable was proved in [24]. A consequence of this effective correspondence is the decidability of equivalence for linear-size increase macro tree transducers. It was indeed shown in [25] that MSO-transducers of ranked trees have decidable equivalence problem (see [34] for a survey on equivalence problems for tree transducers).

Other connections between MSO-transducers and tree transducers have been obtained, for an extension of streaming string transducers to trees [4], and for some classes of tree walking transducers [16].

Acknowledgments I warmly thank Jean-François Raskin and Jean-Marc Talbot for reading preliminary versions of this paper, and Sebastian Maneth for his helpful comments.

References

1. A. Alur and P. Černý. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *P*OPL, pages 599–610, 2011.
2. R. Alur and P. Černý. Expressiveness of streaming string transducers. In *F*STTCS, volume 8, pages 1–12, 2010.
3. R. Alur, E. Filiot, and A. Trivedi. Regular transformations of infinite strings. Technical Report MS-CIS-12-05, University of Pennsylvania, 2012.
4. R. Alur and L. D'Antoni. Streaming tree transducers. In *I*CALP, volume 7392 of *L*NCS, pages 42–53. Springer, 2012.
5. R. Alur and J.V. Deshmukh. Nondeterministic streaming string transducers. In *I*CALP, volume 6756 of *L*NCS, pages 1–20. Springer, 2011.
6. R. Alur, A. Durand-Gasselin, and Ashutosh Trivedi. From monadic second-order definable string transformations to transducers. In *L*ICS, pages 458–467, 2013.
7. M.-P. Béal, O. Carton, C. Prieur, and J Sakarovitch. Squaring transducers: an efficient procedure for deciding functionality and sequentiality. *T*heoretical Computer Science, 292(1):45–63, 2003.
8. R. Bloem and J. Engelfriet. A comparison of tree transductions defined by monadic second order logic and by attribute grammars. *J.* Comput. Syst. Sci., 61(1):1–50, 2000.
9. M. Bojanczyk. Transducers with origin information. In *I*CALP, pages 26–37, 2014.

10. J. R. Büchi. Weak second-order arithmetic and finite automata. *Z*eitschrift für Mathematische Logik und Grundlagen der Mathematik, 6(1–6):66–92, 1960.
11. J. R. Büchi. On a decision method in restricted second-order arithmetic. In *I*nt. Congr. for Logic Methodology and Philosophy of Science, pages 1–11. Standford University Press, Stanford, 1962.
12. A. Church. An unsolvable problem of elementary number theory. *A*mer. J. Math., 58:345–363, 1936.
13. H. Comon-Lundh, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *T*ree Automata Techniques and Applications. November 2007.
14. B. Courcelle. Monadic second-order definable graph transductions: a survey. *T*heoretical Computer Science, 126(1):53–75, 1994.
15. B. Courcelle. Monadic second-order definable graph transductions: A survey. *T*heoretical Computer Science, 126:53–75, 1994.
16. B. Courcelle and J. Engelfriet. *G*raph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach, volume 138 of *E*ncyclopedia of mathematics and its applications. Cambridge University Press, 2012.
17. K. Culik II and J. Karhumäki. The equivalence problem for single-valued two-way transducers (on NPDTOL languages) is decidable. *S*IAM J. Comput., 16(2):221–230, 1987, April.
18. H. B. Curry. Functionality in Combinatory Logic. In *P*roceedings of the National Academy of Sciences of the United States of America, volume 20, pages 584–590, November 1934.
19. V. Diekert and P. Gastin. First-order definable languages. In *L*ogic and Automata: History and Perspectives, Texts in Logic and Games, pages 261–306. Amsterdam University Press, 2008.
20. H. D. Ebbinghaus and J. Flum. *F*inite Model Theory. Springer-Verlag, Berlin, 1995.
21. C. C. Elgot. Decision problems of finite automata design and related arithmetics. *I*n Transactions of the American Mathematical Society, 98(1):21–51, 1961.
22. J. Engelfriet and H. J. Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *A*CM Trans. Comput. Logic, 2:216–254, 2001.
23. J. Engelfriet and S. Maneth. Macro tree transducers, attribute grammars, and MSO definable tree translations. *I*nf. Comput., 154(1):34–91, 1999.
24. J. Engelfriet and S. Maneth. Macro tree translations of linear size increase are MSO definable. *SIAM* J. Comput., 32(4):950–1006, 2003.
25. J. Engelfriet and S. Maneth. The equivalence problem for deterministic MSO tree transducers is decidable. *I*nf. Process. Lett., 100(5):206–212, 2006.
26. E. Filiot, O. Gauwin, P.A. Reynier, and F. Servais. From two-way to one-way finite state transducers. In *L*ICS, pages 468–477, 2013.
27. E. Filiot, S.N. Krishna, and A. Trivedi. First-order definable string transformations. 2014. To appear in FSTTCS.
28. Z. Fülöp and H. Vogler. *S*yntax-Directed Semantics - Formal Models Based on Tree Transducers. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 1998.
29. T. V. Griffiths. The unsolvability of the equivalence problem for -free nondeterministic generalized machines. *J*ournal of the ACM, 1968.
30. W. Howard. The formulae-as-types notion of construction. In *T*o H.B.Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism. Academic Press., 1980.

31. N. Immerman. Languages that capture complexity classes. *S*IAM Journal on Computing, 16(4):760–778, 1987.
32. J. Berstel. *T*ransductions and Context-Free Languages. Teubner, Stuttgart, 1979.
33. C. Lautemann, P. McKenzie, T. Schwentick, and H. Vollmer. The descriptive complexity approach to LOGCFL. *J*ournal of Computer and System Sciences, 62, 2001.
34. S. Maneth. Equivalence problems for tree transducers: A brief survey. In *A*FL, pages 74–93, 2014.
35. P. McKenzie, T. Schwentick, D. Therienand H. Vollmer. The many faces of a translation. *J*ournal of Computer and System Sciences, 72, 2006.
36. R. McNaughton. Testing and generating infinite sequences by a finite automaton. *I*nformation and Control., 9:521–530, 1966.
37. M.O. Rabin. Decidability of second-order theories and automata on infinite trees. *T*rans. Amer.Math.Soc., 141:1–35, 1969.
38. J. Sakarovich. *E*lements of Automata Theory. Cambridge University Press, Cambridge, England, 2009.
39. M.P. Schützenberger. On finite monoids having only trivial subgroups. *I*nformation and Control, 8(2):190–194, April 1965.
40. J.C. Shepherdson. The reduction of two-way automata to one-way automata. In *E*dward F. Moore (Ed.), Sequential Machines: Selected Papers, Addison-Wesley. 1964.
41. H. Straubing. *F*inite Automata, Formal Logic, and Circuit Complexity. Birkhäuser, Boston, Basel and Berlin, 1994.
42. W. Thomas. Languages, automata and logic. In A. Salomaa and G. Rozenberg, editors, *H*andbook of Formal Languages, volume 3, Beyond Words. Springer, Berlin, 1997.
43. B.A. Trakhtenbrot. Finite automata and logic of monadic predicates (in Russian). *D*okl. Akad. Nauk SSSR, 140:326–329, 1961.
44. Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *P*rocedings of the London Mathematical Society, 42(2):230–265, 1936.
45. M.Y. Vardi and T. Wilke. Automata: from logics to algorithms. In *L*ogic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]., pages 629–736, 2008.