Regular Transformations of Data Words Through Origin Information

Antoine Durand-Gasselin[1] and Peter Habermehl[2(✉)]

[1] Aix Marseille Université, CNRS, Centrale Marseille, LIF UMR 7279, Marseille, France
Antoine.Durand-Gasselin@centrale-marseille.fr
[2] IRIF, Univ. Paris Diderot & CNRS, Paris, France
Peter.Habermehl@liafa.univ-paris-diderot.fr

Abstract. We introduce a class of transformations of finite data words generalizing the well-known class of regular finite string transformations described by MSO-definable transductions of finite strings.These transformations map input words to output words whereas our transformations handle data words where each position has a letter from a finite alphabet and a data value. Each data value appearing in the output has as origin a data value in the input. As is the case for regular transformations we show that our class of transformations has equivalent characterizations in terms of deterministic two-way and streaming string transducers.

1 Introduction

The theory of transformations of strings (or words) over a finite alphabet has attracted a lot of interest recently. Courcelle [8] defined finite string transformations in a logical way using Monadic second-order definable graph transductions. Then, a breakthrough was achieved in [9] where it was shown that these transformations are equivalent to those definable by deterministic two-way finite transducers on finite words. In [1] deterministic streaming string transducers (SST) on finite words were introduced. This model is one-way but it is equipped with string variables allowing to store words. It is equivalent [1] to the deterministic two-way finite transducers and to MSO-definable transformations. Interestingly, the motivation behind SST was the more powerful SDST model [2]. SDST work on data words, i.e. words composed of couples of letters from a finite alphabet and an infinite data domain. However, they do not have the same nice theoretical properties as SST, for example they are not closed under composition because SDST have data variables allowing to store data values and *compare* data values with each other. Furthermore, there is no equivalent logical characterization.

This work was supported in part by the VECOLIB project (ANR-14-CE28-0018) and by the PACS project (ANR-14-CE28-0002).

A. Durand-Gasselin—Part of this work was done while this author was at Technical University Munich.

B. Jacobs and C. Löding (Eds.): FOSSACS 2016, LNCS 9634, pp. 285–300, 2016.
DOI: 10.1007/978-3-662-49630-5_17

In this paper, analogously to the case of string transformations of finite words, we obtain a class of transformations of finite data words which has an MSO characterization as well as equivalent characterizations in terms of deterministic two-way transducers and streaming transducers. To achieve this, we allow storing of data values in the transducers but not comparison.

As an example we consider the transformation taking a finite input word over $\{\#, a, b\}$ starting and finishing with a #, together with associated data values from the integers, like $\binom{\#ab\#abb\#}{1\,24\,5\,671\,4}$ and produces as output the word where (1) #'s are left unchanged, and between successive #'s (2) words w in $\{a, b\}^*$ are transformed into $w^R w$ where w^R denotes the reverse image of w, and (3) the data value associated to each a is the value of the first # and the value for b's is the value of the second #. So, the output for the example word is $\binom{\#baab\#bbaabb\#}{1\,5115\,5\,445544\,4}$.

It is clear how a deterministic two-way transducer with the ability of storing data values can realize this transformation: it stores the first data value (1) in a *data variable* while outputting $\binom{\#}{1}$, then goes to the second #, stores the corresponding data value (5) in a second data variable, and goes back one by one while producing $\binom{ba}{51}$. Then, it turns around at the first #, goes again to the second # producing $\binom{ab}{15}$ and restarts the same process.

Now, to realize this transformation with a deterministic streaming string transducer one has to make with the fact that they can only go through the input word once from left to right. Nevertheless we will introduce a model which can realize the described transformation: in between two $\#'s$ it stores the so-far read string and its reverse in a *data word variable*. As the data value of the second $\#'$ is not known in the beginning it uses a *data parameter* p instead. For example, before the second #, the stored word will be $\binom{baab}{p11p}$. When reading the second #, it then replaces p everywhere by 5 and stores the result in another data word variable. The same repeats for the rest of the word until the end is reached and the output contained in a data word variable.

The same transformation can also be described logically. To define transformations on data words, a natural choice would be to use transducers with origin information and their corresponding MSO transductions studied in [6]. Basically, their semantics also takes into account information about the origin of a letter in the output, i.e. the position in the input from which it originates. Obviously, this can be generalized to data values by defining the data value of each output position as the data value in the input position from where the output originated. This definition is however not expressive enough to handle our example, since an input position can only be the origin of a bounded number of output positions but the data values attached to (unboundedly many) a's and b's between two successive #'s come from the same two input positions.

Therefore, in this paper, we first introduce a logical characterization of word transformations with generalized origin information. Our logical characterization is an extension of classical MSO transductions with an additional functional MSO defined relation that maps each element of the interpretation (symbols of the output word) to an element of the interpreted structure (symbols of the input word). This generalization naturally defines transformations of data words;

the data value at a given position of the output is the data value carried at the corresponding position in the input. This suffices to define the previously described example transformation.

Interestingly, our class of transformations is captured by a natural model of deterministic two-way transducers extended with data variables whose values can neither be tested nor compared. By adding data word variables (as in streaming string transducers) containing data values and parameters, we then manage, while preserving determinism, to restrict that model to a one-way model. Data parameters are placeholders for data values which can be stored in data word variables and later replaced by concrete data values. We show that this one-way model can be captured by MSO to achieve the equivalence of all three models.

2 MSO Interpretations with MSO Origin Relation

2.1 Words, Strings and Data Words

For S a set of symbols, we denote by S^* the set of finite words (i.e. the set of finite sequences of elements of S) over S. Given a word w, we can refer to its length ($|w|$), its first symbol ($w[0]$), the symbol at some position $i < |w|$ in the word ($w[i]$), some of its subwords (e.g. $w[i{:}j]$ with $0 \leq i \leq j < |w|$, the subword between positions i and j) etc. In this paper, we only consider finite words.

An alphabet (typically denoted Σ or Γ) is a finite set of symbols. Furthermore, we use a (potentially infinite) set of *data values* called Δ. In the sequel, we use *string* to refer to a (finite) word over a finite alphabet and *data word* to refer to a word over the cartesian product of a finite alphabet and a set of data values. Since symbols of data words consist of a letter (from the finite alphabet) and a data value, we can naturally refer to the data value at some position in a data word, or the string corresponding to some data word. Conversely a string together with a mapping from its position to Δ forms a data word.

A string w (over alphabet Σ) is naturally seen as a directed node-labeled graph (rather than considering edges only connecting two successive positions, we take the transitive closure: thus the graph is a finite linear order). The graph is then seen as an interpreted relational structure whose domain is the positions of w, with a binary edge predicate $<$, and a monadic predicate for each letter of Σ. We denote $\mathcal{S}_\Sigma$ the signature consisting of $<_{/2}$ and $\sigma_{/1}$ for each $\sigma \in \Sigma$.

Any string over alphabet Σ is an interpretation over a finite domain of $\mathcal{S}_\Sigma$, conversely any interpretation of $\mathcal{S}_\Sigma$ is a string if (1) its domain is finite, (2) $<$ defines a linear order and (3) at every position exactly one monadic predicate holds. We remark that (2) and (3) can be expressed as monadic second order (MSO) sentences. Two interpretations are isomorphic iff they are the same string.

With this logic based approach we have a very simple classical characterization of regular languages: a language L over alphabet Σ is regular iff there exists an MSO sentence φ (over signature $\mathcal{S}_\Sigma$) such that the set of interpretations of $\mathcal{S}_\Sigma$ with finite domain satisfying φ is the set of strings in L.

2.2 MSO Interpretations

Using this model theoretic characterization of strings, we can define a class of transformations of strings. For the sake of clarity we consider transformations of strings over alphabet Σ to strings over alphabet Γ. We now define an MSO interpretation of $\mathcal{S}_\Gamma$ in $\mathcal{S}_\Sigma$, as $|\Gamma|+2$ MSO formulas over signature $\mathcal{S}_\Sigma$: $\varphi_<$ with two free first-order variables and φ_{dom} and $(\varphi_\gamma)_{\gamma\in\Gamma}$ with one free first-order variable. Any interpretation $\mathcal{I}_\Sigma$ (of the signature $\mathcal{S}_\Sigma$) defines an interpretation of the structure $\mathcal{S}_\Gamma$: its domain is the set of elements of the domain of $\mathcal{I}_\Sigma$ satisfying φ_{dom} in $\mathcal{I}_\Sigma$, and the interpretation of the predicates over that domain is given by the truth value of each of the other MSO formulas.

An important remark is that if the interpretation $\mathcal{I}_\Sigma$ has finite domain, then so will the constructed interpretation of $\mathcal{S}_\Gamma$. Also, since we can express in MSO (over the signature $\mathcal{S}_\Gamma$) that the output structure is a string (with (2) and (3)), we can also express in MSO over the signature $\mathcal{S}_\Sigma$ that the output structure is a string, hence we can decide whether for any input string our MSO interpretation produces a string.

Above, we presented the core idea of Courcelle's definition [8] of MSO graph transductions. Courcelle further introduces the idea of interpreting the output structure in several copies of the input structures. To define such a transduction, we need to fix a *finite* set C of copies, the domain of the output structure will thus be a subset of the cartesian product of the domain of the input structure with the set of copies. The transduction consists of $|C|^2+(|\Gamma|+1)|C|+1$ MSO formulas over the input structure:

- the sentence φ_{indom} that states whether the input structure is in the input domain of the transduction,
- formulas φ^c_{dom} (with one free first-order variable) for each c in C, each stating whether a node x in copy c is a node of the output structure,
- formulas φ^c_γ (also with one free first-order variable), for each $c\in C$ and each $\alpha\in\Gamma$ which states whether a node x in copy c is labelled by α,
- and formulas $\varphi^{c,d}_<$ (with two free first-order variables, namely x,y) that states whether there exists an edge from x in copy c to y in copy d.

The semantics of these transformations naturally provides a notion of origin: by definition a node of the output structure is a position x in the copy c of the input structure (such that $\varphi^c_{dom}(x)$ is true).

2.3 Transduction of Data Words

Data words cannot be represented as finite structures (over a finite signature) but they can be seen as strings together with a mapping of positions to data values.

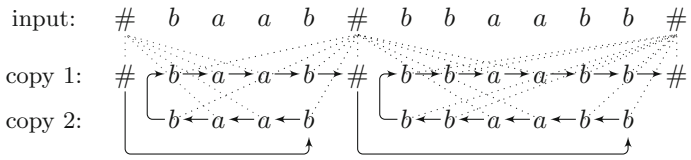
To define a data word transduction, we take a string transduction that we extend with an MSO relation between positions in the input word and positions in the output word. Formally we extend the definition of MSO transduction with $|C|$ MSO formulas (with two free first-order variables) $\varphi^c_{orig}(x,y)$, which we call

the origin formulas, stating that position x in copy c (in the output string) corresponds to position y in the input string. We further impose that for any input word in the domain of the transformation and any x and $c \in C$ such that x in copy c is in the output of the transformation, there exists exactly one y that validates $\varphi^c(x, y)$. We remark that this restriction can be ensured in MSO (over the input structure). Then, the data value at each output position is defined to be the data value at the corresponding input position.

We call MSOT the class of string transformations defined as MSO interpretations, and MSOT+O the class of data word transformation defined as MSO interpretations together with origin formulas. We remark that this definition of origin captures the usual origin information in the sense of [6] by fixing $\varphi^c_{orig}(x, y) \equiv (x = y)$.

2.4 The Running Example

Two copies suffice to define the transformation for the running example. For clarity, we do not represent the ordering relation $<$, but rather the successor relation.



φ_{indom} states the input word starts and ends with a #. $\varphi^1_{dom}(x)$ is true (every node in the first copy is part of the output), while $\varphi^2_{dom}(x) = \neg\#(x)$ tests the letter in the input at that position is not a #. The labeling formulas are the identity ($\varphi^1_a(x) = a(x)$,...) —the behaviour of the formula outside the output domain is considered irrelevant. $\varphi^{1,1}_<(x, y) = x < y$, and $\varphi^{2,2}_<(x, y)$ checks if there is a #-labeled position between position x and y (in the input): if so it ensures that $x < y$, if not it ensures $x > y$. $\varphi^{1,2}(x, y)$ and $\varphi^{1,2}(x, y)$ also distinguish cases whether there is a #-labeled position between x and y or not.

The origin information MSO formulas happen here to be the same for the two copies $\varphi^i(x, y)$ making cases on the letter x: if it is an a (resp. a b) it ensures y is the first #-labeled position before (resp. after) position x.

2.5 Properties

Defining word transformations through MSO interpretations yields some nice properties:

Theorem 1. *MSOT+O is closed under composition.*

Proof. MSOT is naturally closed under composition: given 2 mso transductions T_1 and T_2, (using C_1 and C_2 copies) we can define $T_1 \circ T_2$ as the MSO-interpretation T_1 of the MSO-interpretation T_2 of the original structure, which is an MSO-interpretation over $C_1 \times C_2$ copies.

In order to show the compositional closure of MSOT+O, it now suffices to define the origin information for the composition of two transductions T_1 and T_2 in MSOT+O. It is clear how to define formulas φ^c_{orig} that relate a position in the output with a position in the input, from the origin formulas of T_1 and T_2. We just need to show these origin formulas are functional; a fact that we easily derive from the functionality of the origin formulas of T_1 and T_2.

The (MSO)-typechecking problem of a transformation is defined as follows:

INPUT: Two MSO sentences $\varphi_{pre}, \varphi_{post}$ and an MSOT+O transformation T
OUTPUT: Does $w \models \varphi_{pre}$ imply that $T(w) \models \varphi_{post}$?

It consists in checking whether some property on the input implies a property on the output, those properties are here expressed in MSO.

Theorem 2. *MSO-typechecking of MSOT+O is decidable.*

Proof. An MSO formula can not reason about data values. Therefore it is sufficient to show that MSO-typechecking of MSOT is decidable. Since the output is defined as an MSO interpretation of the input, it is easy to convert an MSO formula on the output into an MSO formula on the input. We just need to check whether the input property implies that converted output property, on any input word, which is checking the universality of an MSO formula over finite strings.

2.6 MSO k-types

Since we present a generalisation of the classical MSO string transductions, the machine models that are expressively equivalent to our logical definition will be extensions of the classical machine models.

To show later that these logical transformations are captured by finite state machines, we use the notion of MSO k-types. We crucially use this notion (more precisely Theorem 3) to prove in Sect. 3 that we only need a finite number of data variables (Lemma 1) to store data values originating from the input.

Given a string w, we define its k-type as the set of MSO sentences of quantifier depth at most k (i.e. the maximum nesting of quantifiers is at most k) that hold for w. A crucial property is that the set of MSO k-types (which we denote Θ_k) is finite and defines an equivalence relation over strings which is a monoïd congruence of finite index. We refer the reader to [11] for more details.

These k-indexed congruences satisfy the following property: two k-equivalent strings will satisfy the same quantifier depth k MSO sentence.

We can extend this notion to MSO formulas with free first-order variables.

Theorem 3. *Given two strings w_1 and w_2 each with two distinguished positions x_1, y_1 and x_2, y_2. $(w_1, (x_1, y_1))$ and $(w_2, (x_2, y_2))$ satisfy the same MSO formulas with quantifier depth at most k and two free first order variables if:*

- *$w_1[x_1] = w_2[x_2]$ and $w_1[y_1] = w_2[y_2]$*
- *x_1, y_1 and x_2, y_2 occur in the same order in w_1 and w_2 (with the special case that if $x_1 = y_1$, then $x_2 = y_2$).*

– *The k-types of the two (strict) prefixes are the same, and the k-types of the two (strict) suffixes are the same, as well as the k-types of the two (strict) subwords between the two positions.*

Proof. Immediate with Ehrenfeucht-Fraïssé games.

3 Two-Way Transducers on Data Words

Two-way deterministic transducers on strings are known to be equivalent to MSO string transductions [9]. Since we process data words and output data words, we will naturally extend this model with a finite set of *data variables*. Notice that the data values in the input word do not influence the finite string part of the output. Therefore the transition function of the transducer may not perform any test on the values of those data variables. However the output word will contain some (if not all) data values of the input word, therefore the model may store some data value appearing in the input word in some variable, and when an output symbol is produced, this is done (deterministically) by combining some letter of the output alphabet together with the data value contained in some data variable.

We start by defining the classical two-way deterministic finite-state transducers (2dft) (with input alphabet Σ and output alphabet Γ) as a deterministic two-way automaton whose transitions are labeled by strings over Γ. The image of a string w by a 2dft A is defined (if w admits a run) as the concatenation of all the labels of the transitions along that run.

Definition 1. *A 2dft is a tuple $(\Sigma, \Gamma, Q, q_0, F, \delta)$ where:*

– *Σ and Γ are respectively the finite input and output alphabets ($\vdash, \dashv \notin \Sigma$)*
– *Q is the finite set of states, q_0 the initial state, and F the set of accepting states*
– *$\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to Q \times \{+1, -1\} \times \Gamma^*$ is the (two-way, Γ^*-labeled) transition function*

A (finite) run of a 2dft A over some string w is a finite sequence ρ of pairs of control states Q and positions in $[-1, |w|]$ (where -1 is supposed to be labeled by $\vdash$ and $|w|$ by $\dashv$), such that: $\rho(0) = (0, q_0)$, $\rho(|\rho|-1) \in \mathbb{N} \times F$ and at any position $k < |\rho| - 1$ in the run, if we denote $\rho(k) = (i_k, q_k)$ and $\rho(k+1) = i_{k+1}, q_{k+1}$, we have that $\delta(q_k, w(i_k)) = (q_{k+1}, i_{k+1} - i_k, u_{k+1})$ for some $u_{k+1} \in \Gamma^*$. Informally $+1$ corresponds to moving to the right in the input string and -1 to moving to the left. The output of A over w is simply the string $u_1 u_2 \dots u_{|\rho|-1}$. We denote $T(A)$ the (partial) transduction from Σ^* to Γ^* defined by A.

Notice that not every input string admits a finite run (since the transducer might get stuck or loop), but if w admits a finite run, it is unique and has length at most $|Q|(|w|+2)$, as this run visits any position at most $|Q|$ times. Therefore a run can also be defined as a mapping from positions of $\vdash w \dashv$ to $Q^{\leq |Q|}$ (sequences of states of length at most $|Q|$).

The next theorem states the equivalence between transformations defined by this two-way machine model and the logical definition of string transformations.

Theorem 4 [9]. *Any string transformation from Σ^* to Γ^* defined by a 2dft can be defined as an MSO interpretation of Γ^* in Σ^* and vice versa.*

Now we define our two-way machine model, *two-way deterministic finite-state transducer with data variables* (2dftv) for data word transformations. We simply extend the 2dft by adding some data variables whose values are deterministically updated at each step of the machine.

Definition 2. *A 2dftv is a tuple $(\Sigma, \Gamma, \Delta, Q, q_0, F, V, \mu, \delta)$ where:*

- *Σ and Γ are respectively the input and output alphabets ($\vdash, \dashv \notin \Sigma$),*
- *Δ is the (infinite) data domain,*
- *Q is the finite set of states, q_0 the initial state, and F the set of accepting states,*
- *V a finite set of data variables with a designated variable $curr \in V$,*
- *$\mu : Q \times \Sigma \times (V \setminus \{curr\}) \to V$ is the data variable update function,*
- *$\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to Q \times \{+1, -1\} \times (\Gamma \times V)^*$ is the (two-way, $(\Gamma \times V)^*$-labeled) transition function.*

We can define the semantics of a 2dftv like the semantics of an 2dft by extending the notion of run. Here, a run is labeled by positions and states but also by a valuation of the variables, i.e. a partial function β which assigns to variables from V values from Δ. This partial function is updated in each step (while reading a symbol different from the endmarkers $\vdash$ or $\dashv$) according to μ and additionally to the variable *curr* the current data value in the input is assigned. The output is obtained by substituting the data variables appearing in the label of the transition relation by their value according to β which we suppose to be always defined (this can be checked easily). Then, naturally a 2dftv defines a transduction from words over $\Sigma \times \Delta$ to words over $\Gamma \times \Delta$.

We call 2DFTV the class of all data word transductions definable by a 2dftv.

Theorem 5. *MSOT+O is included in 2DFTV.*

The challenge to show the theorem is to be able to extend the MSOT to 2DFT proof from [9], so as to be able to also carry in data variables all the necessary data values needed in the output.

We recall the key features in the proof of [9]. First, 2dft's are explicitly shown to be composable [7], which gives regular look-around (the ability to test if the words to the left and to the right of the reading head are in some regular languages) for free: a first pass rewrites the input right-to-left and adds the regular look-ahead, and the second pass re-reverses that word while adding the regular look-back. It is then possible (by reading that regular look-around) to implement MSO-jumps. Given an MSO formula φ with 2 free variables, an MSO-jump φ from position x consists in directly going to a position y such that $\varphi(x, y)$ holds. Using MSO-jumps 2dft can then simulate MSO transformations.

We show thereafter how to extend such a 2dft that takes as input the (look-around enriched) string and produces its image, to a 2dftv. The proof is then

in three steps: first we show that a finite number of data variables is needed, then we briefly describe how to update those data variables: each transition of the 2dft being possibly replaced by a "fetching" of exactly one data variable, and finally it is easy to see how to compose the preprocessing 2dft with that produced 2dftv.

To store only a finite number of data values, we will only store those which originate from a position on one side of the currently processed position and that are used on the other side of the currently processed position. The following lemma ensures a bound on the number of data variables.

Lemma 1. *Let w be a data word, x a position in w, and T a transducer. Denote k the quantifier depth of origin formulas. There are at most $|\Sigma||\Theta_k|^2$ positions $z > x$ such that there exists a position $y < x$ in some copy c such that $\varphi^c_{orig}(y, z)$ holds, i.e. that the data value carried by y in copy c is that of z.*

Proof. By contradiction, we use the pigeon hole principle. We can find two distinct positions z and z' such that the type of the subword between x and z and x and z' is the same, and the type of the suffix from z is the same as the type of the suffix from z'.

Let y a position, left of x where the data value of z is used, thus $\varphi^c_{orig}(y, z)$ holds. We apply Theorem 3 to $(w, (y, z))$ and $(w, (y, z'))$ and therefore $\varphi^c_{orig}(y, z')$ also holds, which contradicts the functionality of the relation φ_{orig}. □

It seems appropriate to name our data variables using MSO types. The data variables are thus $\Sigma \times \Theta_k \times \Theta_k \times \{l, r\}$, $(\sigma, \tau_1, \tau_2, l)$ denoting the data variable containing the data value from the position y (in the input word which is labeled by σ), left (l) of current positions, such that the prefix up to y has type τ_1, and the subword between y and current position has type τ_2.

With an appropriate value of k' (greater than k) the knowledge of the k'-types of the prefix and suffix of the word from the currently processed position, informs us for each data variable whether it contains a value or not, whether it is used at the current position and most importantly to which data variable the value should be transfered when a transition to the right (or the left) is taken.

Notice that when the 2dft performs a transition to the right, four things can happen (only 2 and 3 are mutually exclusive):

1. A data value from a previous position was used for the last time and should be discarded
2. The current data value has been used earlier and will not be used later (and should be discarded)
3. The current data value may be used later and was not used before (and thus should be stored)
4. A data value from a next position is first used (and thus should be stored)

The challenging part is the case (4), as we would need to fetch the data value which we suddenly need to track. The new value is easily fetched through an MSO jump (to the right) which is a feature introduced by [9] allowing to jump

to a position in the input specified by an MSO formula. In turn this jump is implemented (thanks to the look-around carried by the input word) as a one-way automaton that goes to the right until it reaches the position where the data value is, and a one-way automaton that goes (left) from that position back to the original position. The challenge is to be able to return to the current position. Thanks to our definition, we can also describe an MSO jump that allows the return: if we had to fetch a new data value, it is because it was *first* used at the position we want to jump back to. Such a position can easily be expressed uniquely with an MSO formula from the position we fetched our data value. We remark that we cannot fetch data values on a per-needed basis (an MSO jump to the position where the data is, is possible, but going back with an MSO-jump is not), which indicates we need data variables.

In the 2dft, any transition for which case (4) happens (this information is contained in the look-around) is replaced by two automata that go fetch (and back) that newly needed data value.

Finally we present how this conversion should work on our example. We need to consider 1-types. Θ_1 is 2^Σ: each characterizing exactly which letters are present in the word. This means hundreds of data variables, but at any point for this transformation, no more than 2 data values will be stored. So long as we read a's we should not have fetched the data value of the following #-labeled position. When a b is read, we fetch that data value and then we can return back to our original position: it is the first position (after the last #) in the word that contains a b.

4 One-Way Transducers

4.1 Streaming String Transducers with Data Variables and Parameters

We first define sstvp, i.e. streaming string transducers with data variables and data parameters. They have the features of streaming string transducers [1,2] extended with *data variables* and *data parameters*. Notice that in contrast to the streaming data-string transducers from [2] sstvp can not compare data values with each other.

Intuitively, sstvp read deterministically data words and compute an output data word. They are equipped with data variables which store data values, parameters which are placeholders for data values and data word variables containing data words which in addition to data values can also contain data parameters. These data parameters can be replaced by data values subsequently.

Definition 3. *A sstvp is a tuple* $(\Sigma, \Delta, \Gamma, Q, X, V, P, q_0, v_0, \delta, \Omega)$ *where:*

- Σ *and* Γ *are respectively the input and output alphabets,*
- Δ *is the (infinite) data domain,*
- Q *is the finite set of states and* $q_0 \in Q$ *the initial state,*
- X *is the finite set of data word variables,*

- V *is the finite set of data variables with a designated variable* $curr \in V$,
- P *is the finite set of data parameters* ($P \cap \Delta = \emptyset$),
- $v_0 : X \to (\Gamma \times P)^*$ *is a function representing the initial valuation of the data word variables.*
- δ *is a (deterministic) transition function:* $\delta(q, \sigma) = (q', \mu_V, \mu_X, \mu_P)$ *where:*
  - $\mu_V : (V \setminus \{curr\}) \to V$ *is the update function of data variables,*
  - $\mu_X : X \to (X \cup (\Gamma \times (V \cup P)))^*$, *is the update function of data word variables,*
  - $\mu_P : P \times X \to P \cup V$ *is the parameter assignment function (dependent on the data word variable).*
- $\Omega : Q \to ((\Gamma \times V) \cup X)^*$ *is the partial output function.*

The streaming string transducers of [1,2] were defined by restricting updates to be *copyless*, i.e. each data word variable can appear only once in an update μ_X. Here, we relax this syntactic restriction along the lines of [5] by considering only 1-bounded sstvp's: informally, at any position the content of some data word variable may only occur once in the output. This allows to duplicate the value of some data word variable in two distinct data word variables, but the value of these variables can not be later combined. It is clear that this condition can be checked and a 1-bounded sstvp can be transformed into a syntactically copyless sstvp one [5].

Now, we define the semantics of sstvp. A valuation of data variables β_V for an sstvp is a partial function assigning data values to data variables. A valuation of data word variables β_X is a function assigning words over $\Gamma \times (\Delta \cup P)$ to data word variables. Then, a *configuration* of an sstvp consists of a control state and a valuation of data and word variables (β_V, β_X). The initial configuration is (q_0, β_V^0, v_0), where β_V^0 is the empty function. When processing a position i in the input word in some state q, first $curr$ is set to the data value at that position in the input, then the data word variables are updated according to μ_X, then the data words contained in data word variables are substituted according to μ_P and finally data variables are updated according to μ_V.

Formally, if $\delta(q, a) = (q', \mu_V, \mu_X, \mu_P)$, then from (q, β_V, β_X) at position i with a letter (a, d) one goes to $(q', \beta''_V, \beta'''_X)$ where:

- $\beta''_V = \beta'_V \cdot \mu_V$, where $\beta'_V = \beta_V[curr \mapsto d]$.
- $\beta'_X = \beta_X \cdot \mu_X$

$$\beta''_X(x) = \beta'_X(x)[v \leftarrow \beta'_V(v)]_{v \in V}$$

$$\beta'''_X(x) = \beta''_X(x)\left[p \leftarrow \begin{cases} \mu_P(x,p) \text{ if } \mu_P(x,p) \in P \\ \beta'_V(\mu_P(x,p)) \text{ if } \mu_P(x,p) \in V \end{cases}\right]$$

For each two data word variables x, x', we say that x at position i flows to x' at position $i+1$ if $x \in \mu_X(x')$. The notion of flow can be easily extended by transitivity, the copylessness restriction forbids that the value of some data word variable at some position i flows more than once to some data word variable at position $j > i$. When reaching the end of the input word in a configuration (q, β),

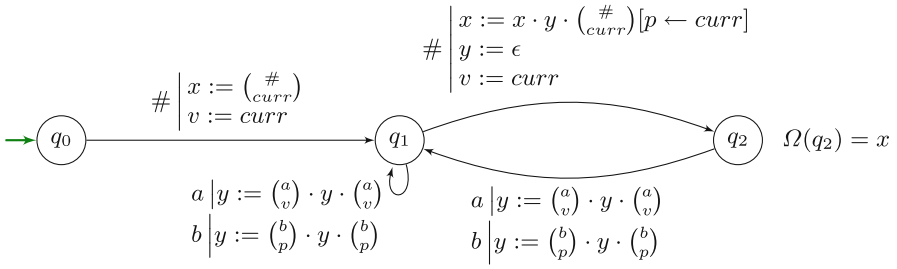


Fig. 1. The sstvp for the running example

a sstvp produces $\beta(\Omega(q))$ if $\Omega(q)$ is defined. Then, naturally a sstvp S defines a transduction from words in $\Sigma \times \Delta$ to words in $\Gamma \times \Delta$.

The sstvp for our running example is given in Fig. 1. All data word variables are initialized with the empty word. By convention, a variable which is not explicitly updated is unchanged. We omit these updates for readability.

Theorem 6. *Equivalence of two sstvp is decidable.*

To prove this theorem we can generalize the proof of decidability of equivalence of SST [2], a reduction to reachability in a non-deterministic one-counter machine. Given two transducers we choose non-deterministically an input string, and one conflicting position in each of the two images (of the transducers): either they are labeled by different letters, or with attached data value originating from two distinct positions in the input word. We keep track in the counter of the difference between the number of produced symbols which will be part of each output before the corresponding conflicting position. Therefore, if the counter reaches 0 at the last letter of the input, the two transducers are different.

We call SSTVP the class of all data word transductions definable by a sstvp.

4.2 From Two-Way to One-Way Transducers

Theorem 7. *2DFTV is included in SSTVP.*

Proof. (Sketch) We use ideas of [1] (based itself on Shepherdson's translation from 2DFA to DFA [10]) where two-way transducers are translated into streaming string transducers. As they translate two-way transducers to copyless streaming string transducers they have to go through an intermediate model called heap-based transducers. Since we relax the copylessness restriction to 1-boundedness we can directly translate 2dftv to sstvp. Furthermore, we have to take care of the contents of data variables of the 2dftv. For that purpose we use data variables and data parameters of the sstvp.

Since an sstvp does only one left-to-right pass on the input word, we cannot revisit any position. As we process a position we need to store all relevant information about that position possibly being later reprocessed by the two-way

transducer. The two-way transducer may process a position multiple times (each time in a different state) each time with a different valuation of data variables and producing some different word: for each state, we need to store in an appropriate data word variable the corresponding production, the valuation of data variables being abstracted with data parameters. Notice that not all these data word variables will be used in the output. Given a 2dftv $A = (\Sigma, \Gamma, \Delta, Q, q_0, F, V, \mu, \delta)$, over which we assume all accepting runs end on the last symbol, we define an sstvp $B = (\Sigma, \Gamma, \Delta, Q', X, V', P, q'_0, v_0, \delta', \Omega)$ as follows:

- $Q' = Q \times [Q \to (Q \times 2^V)]$

  A state of the one-way transducer consists of a state of the two-way transducer and a partial mapping from states to a pair of a state and a set of variables. As a position $i+1$ is processed, the state of B contains the following information: in which state A first reaches position i and for each state q of A what would be the state of A when it reaches for the first time position $i+1$ had it started processing position i from state q: this part is the standard Shepherdson's construction. The function is partial, as from position i from some states A might never reach position $i+1$ (getting stuck).
  We remark that along the subrun from position i (in state q) to position $i+1$, the A might store some data values in some data variables. The set of data variables denotes the set of data variables the two-way transducer has updated along that run.
- $X = x_l \cup \{x_q \mid q \in Q\}$
  At position i, variable x_l will store the word produced by A until it first reaches position i. Variable x_q will store the word produced from position i in state q until position $i+1$ is first reached.
- $V' = V \cup \{v_q \mid v \in V, q \in Q\}$
  At position $i+1$, data variable v will contain the value of variable v of A as it first reaches position $i+1$. Assume that B reaches position i in some state (q, f) with $f(q') = (q'', W)$, and $v \in W$. Then variable $v_{q'}$ will contain the last value stored in v when A processes from position i in state q' until it first reaches position $i+1$.
- $P = \{p_{v,q} \mid v \in V, q \in Q\}$
  At position i, parameter $p_{v,q}$ will be present only in data word variable x_q, representing that along the run of A the data value from data variable v at position i in state q was output before $i+1$ was first reached. Such a symbol needs to be present in x_q, but the data value is not yet known, hence it is abstracted by the data parameter $p_{v,q}$.

It is then easy to see how to define q'_0 and δ' so as to preserve these invariants. As B can not see $\dashv$, B must maintain the possible output in an extra variable, where it is supposed that the next symbol would be $\dashv$.

We now detail an example (see Fig. 2) so as to give an intuition how $\delta'(q, \sigma)$ is built: we will specifically focus on the value of x_{q_1}. We denote f the second component of q and we assume that $f(q_2) = (q_3, \{v_1, v_2\})$, $f(q_4) = (q_5, \{v_2, v_3\})$.

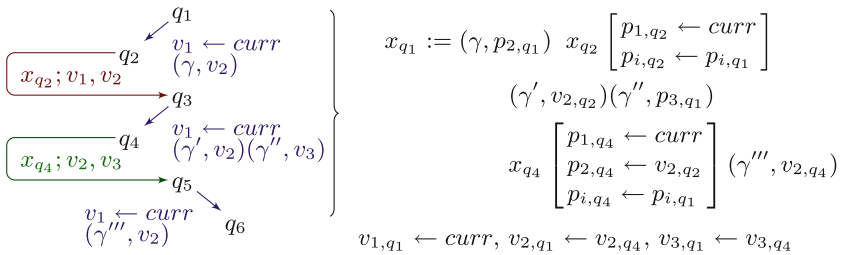


Fig. 2. An example to illustrate the transformation from A to B.

Furthermore, we assume that in A, $\delta(q_1, \sigma) = (q_2, -1, (\gamma, v_2))$ and $\delta(q_3, \sigma) = (q_4, -1, (\gamma', v_2)(\gamma'', v_3))$ and finally that $\delta(q_5, \sigma) = (q_6, +1, (\gamma''', v_2))$. Also reading σ in q_1 and q_3 assigns the current data value to v_1 (i.e. $\mu(q_1, \sigma, v_1) = \mu(q_3, \sigma, v_1) = curr$), other data variables are not modified (i.e. $\mu(q_1, \sigma, v_i) = \mu(q_3, \sigma, v_i) = v_i$).

By the aforementioned invariants, from state q_1, A will first reach the following position in state q_6 (from the σ-labeled position in state q_1, it first goes left, reaches it again in state q_3, goes left again, arrives in state q_5 and then moves to the right in state q_6).

If we abstract the data values, the content of the data word variable x_{q_1} will thus be $(\gamma, ?)x_{q_2}(\gamma', ?)(\gamma'', ?)x_{q_4}(\gamma''', ?)$. Now we detail data attached to the produced letters, and the parameter assignments in the data word variables:

γ will be given the data parameter p_{2,q_1}.

In x_{q_2}: since a data value is assigned to v_1 between q_1 and q_2, p_{1,q_1} should be substituted by that data value (which is $curr$) in x_{q_2}. Other parameters in x_{q_2} (which are all of the form p_{i,q_2}) are substituted by the corresponding p_{i,q_1}.

γ' will be given the data value v_{2,q_2} and (because v_3 has not been assigned a data value since q_1) γ'' will be assigned the data parameter p_{3,q_1}.

In x_{q_4}: as a data value was assigned to v_2 between q_2 to q_3, parameter p_{2,q_4} will be substituted by that value i.e. v_{2,q_2}; parameter p_{1,q_4} will be substituted by $curr$ and all other parameters (which are of the form p_{i,q_4}) will be assigned the corresponding data parameters p_{i,q_1}.

γ''' should be assigned data value v_{2,q_4}.

Therefore by reading a σ in B, we reach a state whose second component maps q_1 to $(q_6, \{v_1, v_2, v_3\})$, $v_{1,q_1} \leftarrow curr$, $v_{2,q_1} \leftarrow v_{2,q_4}$, $v_{3,q_1} \leftarrow v_{3,q_4}$.

4.3 From One-Way Transducers to MSO

In order to conclude that the three models of data word transformations are equivalent, it remains to show that our MSO transductions with MSO origin information capture all transformations defined by the one-way model.

Theorem 8. *SSTVP is included in MSOT+O.*

The proof is very similar to that of encoding finite state automata in MSO. Usually to show that MSO captures string transformations defined by a one-way model one defines an output graph with Γ-labeled edges and ϵ-edges. We directly give a proof that builds a (string) graph whose nodes are Γ-labeled.

Given an sstvp S we fix the set of copies C as the set of occurrences of symbols of Γ in the variable update function.

Since S is deterministic, we will write an MSO sentence φ that characterizes a run of a word in S. This formula will be of the form $\exists X_1, \dots X_n \psi$, such that given a word w (in the domain of the transformation), there exists a unique assignment of the X_i such that ψ holds. These second order variables consist of:

- X_q for $q \in Q$: position $i \in X_q$ iff processing position i yielded state q.
- X_r for every word variable r: position $i \in X_r$ iff the content of variable r will flow in the output
- X_{r_1,r_2} for every pair of distinct word variables r_1, r_2: position $i \in X_{r_1,r_2}$ iff the content of variable r_1 will flow in the output before the content of the variable r_2 that will also flow in the output.

Our sequential machine model allows easily to write such a formula ψ. With the formula ψ, we can write formulas φ_{indom}, $(\varphi^c_{dom})_{c\in C}$, $(\varphi^c_{\gamma})_{c\in C}$, and $(\varphi^{c,d}_{<})_{c,d\in C}$. We remark that second order variables X_{r_1,r_2} have a unique valid assignment because of the (semantic) copylessness of sstvp. These variables are typically used to define $\varphi^{c,d}_{<}$.

To hint how to build formula $\varphi^c_{orig}(x, y)$ we state the following simple lemma about runs of sstvps.

Lemma 2. *Given an sstvp S, an input word w and position x that produces a symbol $\gamma \in \Gamma$ that will be part of the output.*

- *Either γ is produced with a data variable (namely v):*
  In this case, there exists a unique position $y \leq x$ where the data value curr was stored in some data variable and that data variable flows to data variable v at position x.
- *or γ is produced with a data parameter (namely p):*
  In this case, there exists a unique position z such that the data parameter attached to γ is some p_m at position z and that p_m is assigned a variable v_m (or curr) at position z. There exists a unique position $y \leq q$ such that at position y the data value curr was put in some data variable, which flows to a data variable v_m at position z.

The notion of "flow" is easily expressed with ψ and second order existential quantification. The copyless semantics of sstvps ensures that to each (output) symbols, exactly one data value (or equivalently a unique position from the input word) is assigned to. This allows to build MSO formulas φ^c_{orig} that have the desired functional property.

5 Conclusion

Finite string transformation have been generalized to infinite string transformations [5] and tree transformations [3,4]. It would be interesting to extend our results to these settings by adding data values and defining transformations via origin information. Furthermore, it would be interesting to study the pre-post condition checking problem along the lines of [2], i.e. the problem to check that given a transducer is it the case that each input satisfying a pre-condition defined via some automata-model is transformed into an output satisfying a similarly defined post-condition.

References

1. Alur, R., Černý, P.: Expressiveness of streaming string transducers. In: FSTTCS, vol. 8, pp. 1–12 (2010)
2. Alur, R., Černý, P.: Streaming transducers for algorithmic verification of single-pass list-processing programs. In: POPL, pp. 599–610 (2011)
3. Alur, R., D'Antoni, L.: Streaming tree transducers. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part II. LNCS, vol. 7392, pp. 42–53. Springer, Heidelberg (2012)
4. Alur, R., Durand-Gasselin, A., Trivedi, A.: From monadic second-order definable string transformations to transducers. In: 28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS, pp. 458–467. IEEE Computer Society (2013)
5. Alur, R., Filiot, E., Trivedi, A.: Regular transformations of infinite strings. In: Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS, pp. 65–74. IEEE Computer Society (2012)
6. Bojańczyk, M.: Transducers with origin information. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014, Part II. LNCS, vol. 8573, pp. 26–37. Springer, Heidelberg (2014)
7. Chytil, M., Jákl, V.: Serial composition of 2-way finite-state transducers and simple programs on strings. In: Salomaa, A., Steinby, M. (eds.) Automata, Languages and Programming. LNCS, vol. 52, pp. 135–147. Springer, London (1977)
8. Courcelle, B.: Monadic second-order definable graph transductions: a survey. Theoret. Comput. Sci. **126**(1), 53–75 (1994)
9. Engelfriet, J., Hoogeboom, H.J.: MSO definable string transductions and two-way finite-state transducers. ACM Trans. Comput. Log. **2**, 216–254 (2001)
10. Shepherdson, J.C.: The reduction of two-way automata to one-way automata. IBM J. Res. Dev. **3**(2), 198–200 (1959)
11. Thomas, W.: Ehrenfeucht games, the composition method, and the monadic theory of ordinal words. In: Mycielski, J., Rozenberg, G., Salomaa, A. (eds.) Structures in Logic and Computer Science. LNCS, vol. 1261, pp. 118–143. Springer, Heidelberg (1997)