Modular Descriptions of Regular Functions

Paul Gastin(✉)

LSV, ENS Paris-Saclay and CNRS, Université Paris-Saclay, Cachan, France
paul.gastin@lsv.fr

Abstract. We discuss various formalisms to describe string-to-string transformations. Many are based on automata and can be seen as operational descriptions, allowing direct implementations when the input scanner is deterministic. Alternatively, one may use more human friendly descriptions based on some simple basic transformations (e.g., copy, duplicate, erase, reverse) and various combinators such as function composition or extensions of regular operations.

We investigate string-to-string functions (which are ubiquitous). A preprocessing that erases comments from a program, or a micro-computation that replaces a binary string with its increment, or a syntactic fix that reorders the arguments of a function to comply with a different syntax, are all examples of string-to-string transformations/functions. We will discuss and compare various ways of describing such functions.

Operationally, we need to parse the input string and to produce an output word. The simplest such mechanism is to use a deterministic finite-state automaton (1DFA) to parse the input from left to right and to produce the output along the way. These are called *sequential* transducers, or one-way input-deterministic transducers (1DFT), see e.g. [17, Chapter V] or [14]. Transitions are labelled with pairs $a \mid u$ where a is a letter read from the input string and u is the word, possibly empty, to be appended to the output string. Sequential transducers allow for instance to strip comments from a latex file, see Fig. 1. Transformations that can be realized by a sequential transducer are called sequential functions. A very important property of sequential functions is that they are closed under composition. Also, each sequential function f can be realized with a canonical minimal sequential transducer $\mathcal{A}_f$ which can be computed from any sequential transducer $\mathcal{B}$ realizing f. As a consequence, equivalence is decidable for sequential transducers.

With a sequential transducer, it is also possible to increment an integer written in binary if the string starts with the least significant bit (lsb), see Fig. 2 left. On the other hand, increment is not a sequential function when the lsb is on the right. There are two possibilities to overcome this problem.

The first solution is to give up determinism when reading the input string. One-way input-nondeterministic finite-state transducers (1NFT) do not necessarily define functions. It is decidable in PTIME whether a 1NFT defines a

M. Ćirić et al. (Eds.): CAI 2019, LNCS 11545, pp. 3–9, 2019.
https://doi.org/10.1007/978-3-030-21363-3_1

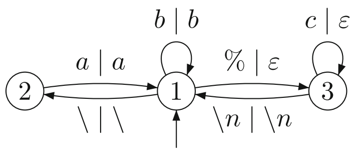


Fig. 1. A sequential transducer stripping comments from a latex file, where $a, b, c \in \Sigma$ are letters from the input alphabet with $b \notin \{\backslash, \%\}$ and $c \neq \backslash n$.

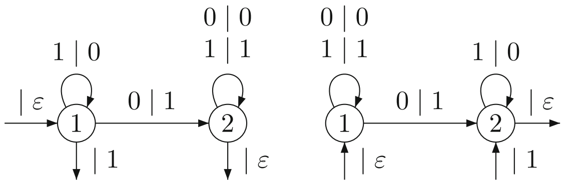


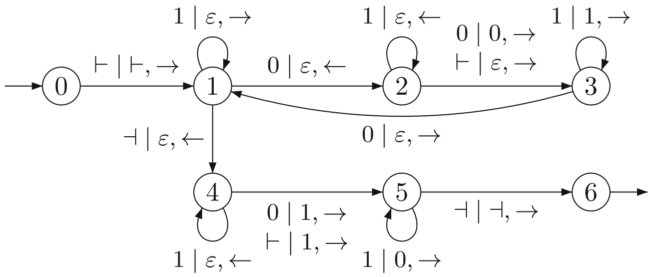
Fig. 2. Transducers incrementing a binary number.

function [18,16]. We are interested in *functional* 1NFT (f1NFT). This is in particular the case when the transducer is input-*unambiguous*. Actually, one-way, input-unambiguous, finite-state transducers (1UFT) have the same expressive power as f1NFT [19]. For instance, increment with lsb on the right is realized by the 1UFT on the right of Fig. 2. Transformations realized by f1NFT are called rational functions. They are easily closed under composition. The equivalence problem is undecidable for 1NFT [15] but decidable in PTIME for f1NFT [18,16]. It is also decidable in PTIME whether a f1NFT defines a sequential function, i.e., whether it can be realized by a 1DFT [7,19].

Interestingly, any rational function h can be written as $r \circ g \circ r \circ f$ where f, g are sequential functions and r is the *reverse* function mapping $w = a_1 a_2 \cdots a_n$ to $w^r = a_n \cdots a_2 a_1$ [12]. We provide a sketch of proof below.[1]

The other solution is to keep input-determinism but to allow the transducer to move its input head in both directions, i.e., left or right (two-way). So we consider two-way input-deterministic finite-state transducers (2DFT) [1]. To realize increment of binary numbers with the lsb on the right with a 2DFT, one has to

[1] Assume that h is realized with a 1UFT $\mathcal{B}$. Consider the unique accepting run $q_0 \xrightarrow{a_1 \mid u_1} q_1 \cdots q_{n-1} \xrightarrow{a_n \mid u_n} q_n$ of $\mathcal{B}$ on some input word $w = a_1 \cdots a_n$. We have $h(w) = u_1 \cdots u_n$. Let $\mathcal{A}$ be the DFA obtained with the subset construction applied to the input NFA induced by $\mathcal{B}$. Consider the run $X_0 \xrightarrow{a_1} X_1 \cdots X_{n-1} \xrightarrow{a_n} X_n$ of $\mathcal{A}$ on w. We have $q_i \in X_i$ for all $0 \leq i \leq n$. The first sequential function f adorns the input word with the run of $\mathcal{A}$: $f(w) = (X_0, a_1) \cdots (X_{n-1}, a_n)$. The sequential transducer $\mathcal{C}$ realizing g is defined as follows. For each state q of $\mathcal{B}$ there is a transition $\delta = q \xrightarrow{(X,a)} p$ in $\mathcal{C}$ if there is a unique $p \in X$ such that $\delta' = p \xrightarrow{a} q$ is a transition in $\mathcal{B}$. Moreover, if δ' outputs u in $\mathcal{B}$ then δ outputs u^r in $\mathcal{C}$. Notice that $q_n \xrightarrow{(X_{n-1}, a_n) \mid u_n^r} q_{n-1} \cdots q_1 \xrightarrow{(X_0, a_1) \mid u_1^r} q_0$ is a run of $\mathcal{C}$ producing $u_n^r \cdots u_1^r = h(w)^r$. The result follows.



0
1
2
3
4
5
6
$\vdash \mid \vdash, \rightarrow$
$1 \mid \varepsilon, \rightarrow$
$0 \mid \varepsilon, \leftarrow$
$1 \mid \varepsilon, \leftarrow$
$0 \mid 0, \rightarrow$
$\vdash \mid \varepsilon, \rightarrow$
$1 \mid 1, \rightarrow$
$0 \mid \varepsilon, \rightarrow$
$\dashv \mid \varepsilon, \leftarrow$
$0 \mid 1, \rightarrow$
$\vdash \mid 1, \rightarrow$
$1 \mid \varepsilon, \leftarrow$
$1 \mid 0, \rightarrow$
$\dashv \mid \dashv, \rightarrow$

Fig. 3. Two-way transducer incrementing a binary number.

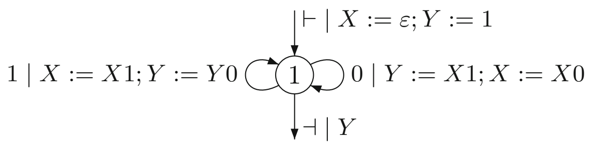
locate the last 0 digit, replace it with 1, keep unchanged the digits on its left and replace all 1's on its right with 0's. This is realized by the 2DFT of Fig. 3. We use $\vdash, \dashv \notin \Sigma$ for the end-markers so the input tape contains $\vdash w \dashv$ when given the input word $w \in \Sigma^*$.

Transformations realized by 2DFTs are called regular functions. They form a very robust class. Regular functions are closed under composition [8]. Actually, a 2DFT can be transformed into a *reversible* one of exponential size [10]. In a reversible transducer, computation steps can be deterministically reversed. As a consequence, the composition of two 2DFTs can be achieved with a single exponential blow-up. Also, contrary to the one-way case, input-nondeterminism does not add expressive power as long as we stay functional: given a f2NFT, one may construct an equivalent 2DFT [13]. Moreover, the equivalence problem for regular functions is still decidable [9].

In classical automata, whether or not a transition can be taken only depends on the input letter being scanned. This can be enhanced using regular look-ahead or look-behind. For instance, the f1NFT on the right of Fig. 2 can be made deterministic using regular look-ahead. In state 1, when reading digit 0, we move to state 2 if the suffix belongs to 1^* and we stay in state 1 otherwise, i.e., if the suffix belongs to $1^*0\{0,1\}^*$. Similarly, we choose to start in the initial state 2 (resp. 1) if the word belongs to 1^* (resp. $1^*0\{0,1\}^*$). More generally, any 1UFT can easily be made deterministic using regular look-ahead: if we have the choice between two transitions leading to states q_1 and q_2, choose q_1 (resp. q_2) if the suffix can be accepted from q_1 (resp. q_2). This query is indeed regular. Hence, regular look-ahead increases the expressive power of one-way deterministic transducers. But regular look-ahead and look-behind do not increase the expressive power of the robust class of regular functions realized by 2DFTs [13].

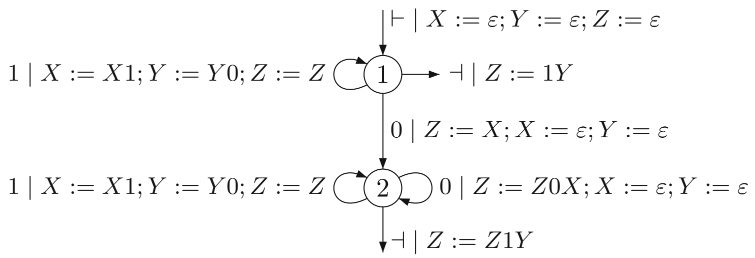
Regular functions are also those that can be defined with MSO transductions [13], but we will not discuss this here.

By using registers, we obtain another formalism defining string-to-string transformations. For instance incrementing a binary number with lsb on the right is realized by the one-way register transducer on Fig. 4. It uses two registers X, Y initialized with the empty string and 1 respectively and updated while reading the binary number. Register X keeps a copy of the binary number read so far, while Y contains its increment. The final output of the transducer



$\vdash \mid X := \varepsilon; Y := 1$
$1 \mid X := X1; Y := Y0$
1
$0 \mid Y := X1; X := X0$
$\dashv \mid Y$

Fig. 4. One-way register transducer incrementing a binary number.



$\vdash \mid X := \varepsilon; Y := \varepsilon; Z := \varepsilon$
$1 \mid X := X1; Y := Y0; Z := Z$
1
$\dashv \mid Z := 1Y$
$0 \mid Z := X; X := \varepsilon; Y := \varepsilon$
$1 \mid X := X1; Y := Y0; Z := Z$
2
$0 \mid Z := Z0X; X := \varepsilon; Y := \varepsilon$
$\dashv \mid Z := Z1Y$

Fig. 5. Streaming string transducer incrementing a binary number.

is the string contained in register Y. This register automaton is a special case of "*simple programs*" defined in [8]. In these simple programs, a register may be reset to the empty string, copied to another register, or updated by appending a finite string. The input head is two-way and most importantly simple programs may be composed. Simple programs coincide in expressive power with 2DFTs [8], hence define once again the class of regular functions.

Notice that when reading digit 0, the transducer of Fig. 4 copies the string stored in X into Y without resetting X to ε. By restricting to one-way register automata with copyless updates (e.g., not of the form $Y := X1; X := X0$ where the string contained in X is duplicated) but allowing concatenation of registers in updates (e.g., $Z := Z0X; X := \varepsilon$), we obtain another kind of machines, called copyless streaming string transducers (SST), once again defining the same class of regular functions [3]. Continuing our example, incrementing a binary number with lsb on the right can be realized with the SST on Fig. 5. It uses three registers X, Y, Z initialized with the empty string and updated while reading the binary number. The final output of the transducer is the string contained in register Z.

The above machines provide a way of describing string-to-string transformations which is not modular. Describing regular functions in such devices is difficult, and it is even more difficult to understand what is the function realized by a 2DFT or an SST. We discuss now more compositional and modular descriptions of regular functions. Such a formalism, called regular list functions, was described in [6]. It is based on function composition together with some natural functions over lists such as reverse, append, co-append, map, etc. Here we choose to look in combinators derived from regular expressions.

The idea is to start from basic functions, e.g., $(1 \mid 0)$ means "read 1 and output 0", and to apply simple combinators generalizing regular expressions [4,2,11,5]. For instance, using the Kleene iteration, $(1 \mid 0)^*$ describes a function which replaces a sequence of 1's with a sequence of 0's of same length. Similarly, $\mathsf{copy} := ((0 \mid 0) + (1 \mid 1))^*$ describes a regular function which simply copies an input binary string to the output. Now, incrementing a binary number with lsb on the right is described with the expression $\mathsf{increment0} := \mathsf{copy} \cdot (0 \mid 1) \cdot (1 \mid 0)^*$, assuming that the input string contains at least one 0 digit. If the input string belongs to 1^*, we may use the expression $\mathsf{increment1} := (\varepsilon \mid 1) \cdot (1 \mid 0)^*$. Notice that such a regular transducer expression (RTE) defines simultaneously the *domain* of the regular function as a regular expression, e.g., $\mathsf{dom}(\mathsf{increment0}) = (0+1)^*01^*$, and the output to be produced. The input regular expression explains how the input should be parsed. If the input regular expression is ambiguous, parsing the input word is not unique and the expression may be non functional. For instance, $\mathsf{copy} \cdot (1 \mid 0)^*$ is ambiguous. The input word $w = 1011$ may be parsed as $10 \cdot 11$ or $101 \cdot 1$ or $1011 \cdot \varepsilon$ resulting in the outputs 1000 or 1010 or 1011 respectively. On the other end, $\mathsf{increment} := \mathsf{increment0} + \mathsf{increment1}$ has an unambiguous input regular expression.

A 2DFT may easily duplicate the input word, defining the function $w \mapsto w\$w$, which cannot be computed with a sequential transducer or a f1NFT. In addition to the classical regular combinators ($+$ for disjoint union, $\cdot$ for unambiguous concatenation or Cauchy product, * for unambiguous Kleene iteration), we add the Hadamard product $(f \odot g)(w) = f(w) \cdot g(w)$ where the input word is read twice, first producing the output computed by f then the output computed by g. Hence the function duplicating its input can be simply written as $\mathsf{duplicate} := (\mathsf{copy} \cdot (\varepsilon \mid \$)) \odot \mathsf{copy}$. This can be iterated duplicating each #-separated words in a string with $f := (\mathsf{duplicate} \cdot (\# \mid \#))^*$. We have $f(u_1\#u_2\#\cdots u_n\#) = u_1\$u_1\#u_2\$u_2\#\cdots u_n\$u_n\#$ when $u_1,\ldots,u_n$ are binary strings.

The Hadamard product also allows to exchange two strings $u\#v \mapsto v\#u$ where $u,v \in \{0,1\}^*$. Let $\mathsf{erase} := ((0 \mid \varepsilon) + (1 \mid \varepsilon))^*$ and

$$\mathsf{exchange} := \Big(\mathsf{erase} \cdot (\# \mid \varepsilon) \cdot \mathsf{copy} \cdot (\varepsilon \mid \#)\Big) \odot \Big(\mathsf{copy} \cdot (\# \mid \varepsilon) \cdot \mathsf{erase}\Big) .$$

Again this can be iterated on the output of the function f defined above with the map

$$g := \mathsf{erase} \cdot (\$ \mid \varepsilon) \cdot (\mathsf{exchange} \cdot (\$ \mid \$))^* \cdot \mathsf{erase} \cdot (\# \mid \varepsilon) .$$

We have $g \circ f(u_1\#u_2\#\cdots u_n\#) = u_2\#u_1\$u_3\#u_2\$\cdots u_n\#u_{n-1}\$$. It turns out that the regular function $g \circ f$ cannot be described using the regular combinators $+$, $\cdot$, $*$, $\odot$. This is the reason for introducing a 2-chained Kleene iteration [4]: $[K,h]^{2+}$ first *unambiguously* parse an input word as $w = u_1u_2\cdots u_n$ with $u_1,\ldots,u_n \in K$ and then apply h to all consecutive pairs of factors, resulting in the output $h(u_1u_2)h(u_2u_3)\cdots h(u_{n-1}u_n)$. For instance, with the functions defined above, we can easily check that $g \circ f = [K,h]^{2+}$ with $K = \{0,1\}^*\#$ and $h := \mathsf{exchange} \cdot (\# \mid \$)$.

Another crucial feature of 2DFTs is the ability to reverse the input $w \mapsto w^r$, e.g., $(1101000)^r = 0001011$. In regular transducer expressions, we introduce a *reversed* Kleene star r-$*$ which parse the input word from left to right but produce the output in reversed order. For instance, $f^{r\text{-}*}(w) = f(u_n)\cdots f(u_2)f(u_1)$

if the input word is parsed as $w = u_1 u_2 \cdots u_n$. Hence, reversing a binary string is described with the RTE $\mathsf{reverse} := ((0 \mid 0) + (1 \mid 1))^{r\text{-}*}$. There is also a reversed version of the two-chained Kleene iteration. With the above notation, we get $[K, h]^{r\text{-}2+}(w) = h(u_{n-1}u_n) \cdots h(u_2 u_3) h(u_1 u_2)$.

Once again, we obtain an equivalent formalism for describing regular functions: the regular transducer expressions using $+$, $\cdot$, $\odot$, $*$, r-$*$, $2+$ and r-$2+$ as combinators [4,2,11,5]. Alternatively, as illustrated on an example above, we may remove the two-chained iterations if we allow function compositions, i.e., using the combinators $+$, $\cdot$, $\odot$, $\circ$, $*$ and r-$*$. Further, we may remove the Hadamard product if we provide $\mathsf{duplicate}$ as a basic function. Indeed, we can easily check that $f \odot g = (f \cdot (\$ \mid \varepsilon) \cdot g) \circ \mathsf{duplicate}$. Also, the reversed Kleene iteration may be removed if we use $\mathsf{reverse}$ as a basic function. We will see that $f^{r\text{-}*} = (f \circ \mathsf{reverse})^* \circ \mathsf{reverse}$. Indeed, assume that an input word is parsed as $w = u_1 u_2 \cdots u_n$ when applying $f^{r\text{-}*}$ resulting in $f(u_n) \cdots f(u_2) f(u_1)$. Then, $\mathsf{reverse}(w)$ is parsed as $u_n^r \cdots u_2^r u_1^r$ when applying $(f \circ \mathsf{reverse})^*$. The result follows since $(f \circ \mathsf{reverse})(u^r) = f(u)$.

To conclude, we have an expressively complete set of combinators $+$, $\cdot$, $*$ and $\circ$ when we allow $\mathsf{duplicate}$ and $\mathsf{reverse}$ as basic functions. We believe that this is a very convenient, compositional and modular, formalism for defining regular functions.

References

1. Aho, A.V., Ullman, J.D.: A characterization of two-way deterministic classes of languages. J. Comput. Syst. Sci. **4**(6), 523–538 (1970)
2. Alur, R., D'Antoni, L., Raghothaman, M.: DReX: a declarative language for efficiently evaluating regular string transformations. In: Rajamani, S.K., Walker, D. (eds.) Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL 2015, pp. 125–137. ACM Press (2015)
3. Alur, R., Deshmukh, J.V.: Nondeterministic streaming string transducers. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6756, pp. 1–20. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22012-8_1
4. Alur, R., Freilich, A., Raghothaman, M.: Regular combinators for string transformations. In: Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS 2014, Vienna, Austria, July 14–18, 2014, pp. 9:1–9:10 (2014)
5. Baudru, N., Reynier, P.-A.: From two-way transducers to regular function expressions. In: Hoshi, M., Seki, S. (eds.) DLT 2018. LNCS, vol. 11088, pp. 96–108. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98654-8_8
6. Bojańczyk, M., Daviaud, L., Krishna, S.N.: Regular and first-order list functions. In: Dawar, A., Grädel, E. (eds.) Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science - LICS 2018, pp. 125–134. ACM Press (2018)
7. Choffrut, Ch.: Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles. Theoret. Comput. Sci. **5**(3), 325–337 (1977)

8. Chytil, M.P., Jákl, V.: Serial composition of 2-way finite-state transducers and simple programs on strings. In: Salomaa, A., Steinby, M. (eds.) ICALP 1977. LNCS, vol. 52, pp. 135–147. Springer, Heidelberg (1977). https://doi.org/10.1007/3-540-08342-1_11
9. Culik, K., Karhumäki, J.: The equivalence of finite valued transducers (on HDT0L languages) is decidable. Theoret. Comput. Sci. **47**, 71–84 (1986)
10. Dartois, L., Fournier, P., Jecker, I., Lhote, N.: On reversible transducers. In: Chatzigiannakis, I., Indyk, P., Kuhn, F., Muscholl, A. (eds.) 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017), volume 80 of Leibniz International Proceedings in Informatics (LIPIcs), pp. 113:1–113:12, Dagstuhl, Germany, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017)
11. Dave, V., Gastin, P., Krishna, S.N.: Regular transducer expressions for regular transformations. In: Hofmann, M., Dawar, A., Grädel, E. (eds.) Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2018), pp. 315–324. ACM Press, Oxford (2018)
12. Elgot, C.C., Mezei, J.E.: On relations defined by generalized finite automata. IBM J. Res. Dev. **9**(1), 47–68 (1965)
13. Engelfriet, J., Hoogeboom, H.J.: MSO definable string transductions and two-way finite-state transducers. ACM Trans. Comput. Log. **2**(2), 216–254 (2001)
14. Filiot, E., Reynier, P.-A.: Transducers, logic and algebra for functions of finite words. SIGLOG News **3**(3), 4–19 (2016)
15. Griffiths, T.V.: The unsolvability of the equivalence problem for lambda-free nondeterministic generalized machines. J. ACM **15**(3), 409–413 (1968)
16. Gurari, E.M., Ibarra, O.H.: A note on finite-valued and finitely ambiguous transducers. Math. Syst. Theory **16**(1), 61–66 (1983)
17. Sakarovitch, J.: Elements of Automata Theory. Cambridge University Press, New York (2009)
18. Schützenberger, M.P.: Sur les relations rationnelles. In: Brakhage, H. (ed.) GI-Fachtagung 1975. LNCS, vol. 33, pp. 209–213. Springer, Heidelberg (1975). https://doi.org/10.1007/3-540-07407-4_22
19. Weber, A., Klemm, R.: Economy of description for single-valued transducers. Inf. Comput. **118**(2), 327–340 (1995)