Boolean Monadic Recursive Schemes as a Logical Characterization of the Subsequential Functions

Siddharth Bhaskar[1](✉), Jane Chandlee[2], Adam Jardine[3], and Christopher Oakden[3]

[1] DIKU, Københavns Universitet, 2100 Copenhagen, Denmark
sbhaskar@di.ku.dk
[2] Haverford College, Haverford, PA, USA
jchandlee@haverford.edu
[3] Rutgers University, New Brunswick, NJ, USA
{adam.jardine,chris.oakden}@rutgers.edu

Abstract. This paper defines boolean monadic recursive schemes (BMRSs), a restriction on recursive programs, and shows that when interpreted as transductions on strings they describe exactly the subsequential functions. We discuss how this new result furthers the study of the connections between logic, formal languages and functions, and automata.

Keywords: Subsequential functions · Logic · Recursive program schemes · Finite automata

1 Introduction

A fundamental result in the connection between automata and logic is that of Elgot [7], Büchi [1], and Trakhtenbrot [21], which states that sentences in monadic second-order (MSO) logic describe exactly the same class of formal languages as finite-state acceptors (FSAs); namely, the regular class of languages. Further work established many connections between restrictions on MSO, restrictions on FSAs, and sub-classes of the regular languages [14,20].

More recently, a major result of Engelfriet and Hoogeboom shows the relationship between MSO and regular *functions* on strings—that is, exactly those functions described by two-way finite state transducers [9]. Essentially, string functions can be described by a MSO *interpretation* in which the binary successor relation and alphabet labels of the output string are defined by a series of binary and unary predicates in the MSO logic of the input strings, relativized over a *copy set* which allows the output string to be larger than the input string. Each element in the output string is thus a copy of an index in the input string, and the character it receives and where it is in the order is determined by which predicates are satisfied by the input string at that index. This technique has

A. Leporati et al. (Eds.): LATA 2020, LNCS 12038, pp. 157–169, 2020.
https://doi.org/10.1007/978-3-030-40608-0_10

allowed a rich study of the relationship between sub-MSO logics and restrictions on finite-state transducers [10,11] parallel to the earlier work on logic and finite-state automata and languages.

However, there remain some interesting classes for which no logical characterization has been previously established. In this paper, we investigate the *subsequential* functions, a strict sub-class of the *rational* functions, or those that are describable by one-way finite-state transducers.[1] While a weak class, there are a number of reasons why the subsequential class is a worthy object of study. From a theoretical perspective, the subsequential functions admit an abstract characterization that generalizes the Myhill-Nerode equivalence classes of regular languages [19]. This property makes the subsequential functions learnable from a sample of positive data [18]. In terms of practical applications, the subsequential functions have applications to speech and language processing [16], and form a hypothesis for the computational upper bound of functions in certain domains of natural language phonology [12,13].

In this paper, we define *boolean monadic recursive schemes* (BMRSs), a restriction on the general notion of a recursive program scheme in the sense of Moschovakis [17]. As indicated by the name, these schemes recursively define a series of unary functions that take as inputs indices from a string and return a boolean value. A system of BMRS functions can be used to express a logical transduction in the sense of Engelfriet and Hoogeboom by assigning these functions to symbols in an output alphabet. An output string then consists of the characters whose functions are true at each index in the input string. We show that string transductions defined by BMRS transductions with predecessor or successor describe exactly the left- and right-subsequential functions, respectively. This is an entirely novel result; the closest result in the literature is that of [6], who give a fragment of least-fixed point logic that captures strict subsets of the left- and right-subsequential functions. As we discuss at the end of the paper, the current result allows for further study of the connections among subclasses of the subsequential functions and of rational functions in general.

This paper is structured as follows. Sections 2 and 3 establish the notation and definitions for strings, subsequential functions, and subsequential transducers. Section 4 establishes BMRSs and BMRS transductions, and Sect. 5 shows the equivalence between BMRS transductions and subsequential functions. Sections 6 and 7 discuss the implications of this result and conclude the paper.

2 Preliminaries

An alphabet Σ is a finite set of symbols; let Σ^* be all strings over Σ, including λ, the empty string. We will frequently make use of special left and right string boundary symbols $\rtimes, \ltimes \notin \Sigma$. We denote by $\rtimes\Sigma^*\ltimes$ the set $\{\rtimes w \ltimes \mid w \in \Sigma^*\}$. Let $\Sigma_{\rtimes} = \Sigma \cup \{\rtimes\}$, likewise $\Sigma_{\ltimes} = \Sigma \cup \{\ltimes\}$ and $\Sigma_{\bowtie} = \Sigma \cup \{\rtimes, \ltimes\}$. For a string w, $|w|$ indicates the length of w. We write w^r for the reversal of w.

[1] We mean *subsequential* in the sense of Schützenberger and Mohri; other authors (e.g. [11]) use the term *sequential* for the same class.

A string u is a *prefix* of w, written $u \sqsubseteq w$, iff $w = uv$ for some string v. For a set $L \subseteq \Sigma^*$ of strings let the *common prefixes* be $\texttt{comprefs}(L) = \bigcap_{w \in L}\{u \mid u \sqsubseteq w\}$. The *longest common prefix* of L is the maximum element in $\texttt{comprefs}(L)$: $\texttt{lcp}(L) = w \in \texttt{comprefs}(L)$ s.t. for all $v \in \texttt{comprefs}(L)$, $|v| \leq |w|$.

3 Subsequential Functions and Transducers

3.1 Abstract Definition

We first define the subsequential functions based on the notion of *tails* [16,19]. Let $f : \Sigma^* \rightarrow \Gamma^*$ be an arbitrary function and $f^p(x) = \texttt{lcp}(\{f(xu) \mid u \in \Sigma^*\})$. Then of course, for every u, $f^p(x) \sqsubseteq f(xu)$. Now let the *tail function* $f_x(u)$ be defined as v such that $f^p(x)v = f(xu)$. This function represents the *tails* of x. This allows us to define the subsequential functions as follows.

Definition 1 (Left-subsequential). *A function f is* left-subsequential *iff the set $\{f_x \mid x \in \Sigma^*\}$ is finite.*

Example 1. For input alphabet $\Sigma = \{a\}$, the function f defined as

$$f(a^n) = \begin{array}{ll} (abb)^{n/2}c & \text{if } n \text{ is even;} \\ (abb)^{(n-1)/2}ad & \text{if } n \text{ is odd,} \end{array}$$

is left-subsequential. Note that for any $x = a^n$ for an even n, $f^p(x) = (abb)^{n/2}$, and so $f_x = f$. For any $y = a^n$ for an odd n, then $f^p(y) = (abb)^{(n-1)/2}a$, and so f_y is the function $f_y(a^m) = d$ if $m = 0$; $bb \cdot f(a^{m-1})$ otherwise. Then f is describable by these two tail functions $\{f_x, f_y\}$.

Conversely, the function g defined as

$$g(a^n) = \begin{array}{ll} ca^{n-1} & \text{if } n \text{ is even;} \\ da^{n-1} & \text{if } n \text{ is odd,} \end{array}$$

is not left-subsequential. Note that for any $x = a^n$, $g^p(x) = \lambda$. This is because $\{g(xu) \mid u \in \Sigma^*\}$ includes both ca^i and da^j for some i and j. Because $g_x(u)$ is defined as v such that $g^p(x)v = g(xu)$, and because $g^p(x) = \lambda$, $g_x(u) = g(xu)$. The consequence of this is that for any $x = a^n$ and $y = a^m$ for a distinct $m \neq n$, for any u, $g_x(u) \neq g_y(u)$. Thus the set of tails functions for g is an infinite set $\{g_{x_1}, g_{x_2}, ...\}$ of distinct functions for each $x_i = a^i$.

The right-subsequential functions are those that are the mirror image of some left-subsequential function.

Definition 2 (Right-subsequential). *A function f is* right-subsequential *iff there is some left-subsequential function f_ℓ such that for any string in the domain of f, $f(w) = (f_\ell(w^r))^r$.*

We leave it to the reader to show that g is right-subsequential. Thus, the subsequential functions are those functions that are either left- or right-subsequential.

3.2 Subsequential Finite-State Transducers

A (left-)subsequential finite-state transducer (SFST) for an input alphabet Σ and an output alphabet Γ is a tuple $\mathcal{T} = \langle Q, q_0, Q_f, \delta, o, \omega \rangle$, where Q is the set of states, $q_0 \in Q$ is the (unique) start state, $Q_f \subseteq Q$ is the set of final states, $\delta : Q \times \Sigma \to Q$ is the transition function, $o : Q \times \Sigma \to \Gamma^*$ is the output function, and $\omega : Q_f \to \Gamma^*$ is the final function. We define the reflexive, transitive closure of δ and o as $\delta^* : Q \times \Sigma^* \to Q$ and $o^* : Q \times \Sigma^* \to \Gamma^*$ in the usual way.

The semantics of a SFST is a transduction $t(\mathcal{T})$ defined as follows; let $t = t(\mathcal{T})$. For $w \in \Sigma^*$, $t(w) = uv$ where $o^*(q_0, w) = u$, and $\omega(q_f) = v$ if $\delta^*(q_0, w) = q_f$ for some $q_f \in Q_f$; $t(w)$ undefined otherwise.

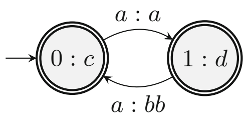

Fig. 1. A graph representation of the SFST for the function f from Example 1.

Theorem 1 ([16,19]). *The left-subsequential functions are exactly those describable by a SFST reading a string left-to-right. The right-subsequential functions are exactly those describable by a SFST reading a string right-to-left and reversing the output.*

Theorem 2 ([16]). *Both the left- and right-subsequential functions are a strict subset of the rational functions.*

For more properties of the subsequential functions and their application to speech and language processing see [16].

4 Boolean Monadic Recursive Schemes

4.1 Syntax and Semantics

We identify strings in $\Sigma^\star$ with structures of the form $\mathbf{S} = \langle D; \sigma_1, \sigma_2, ..., \sigma_n, p, s \rangle$ where the *domain* D is the set of indices; for each character $\sigma \in \Sigma_{\rtimes\ltimes}$, we also write σ_i for the unary relation $\sigma_i \subseteq D$ selecting the indices of that character (and we assume that the least and greatest indices contain the characters $\rtimes$ and $\ltimes$, respectively); p is the predecessor function on indices (fixing the least index); and s is the successor function on indices (fixing the greatest index). As an abbreviatory convention we use $\mathtt{x} - i$ for i applications of p to $\mathtt{x}$, and likewise $\mathtt{x} + i$ for i applications of s. (E.g. $\mathtt{x} - 2$ is the same as $p(p(\mathtt{x}))$).

Boolean monadic recursive schemes are simple programs that operate over such string structures. They are a particular case of the *recursive programs* of Moschovakis [17]. We briefly review the syntax and semantics of such recursive programs in this particular signature, then impose (Definition 3) the pertinent syntactic restriction to obtain BMRSs.

Data and Variables. We have two types of data: boolean values and string indices. We have two countably infinite set of variables: *(index) variables* $\mathtt{X}$, which range over string indices, and *recursive function names* $\mathtt{F}$. Each recursive function name $\mathtt{f} \in \mathtt{F}$ comes with an *arity* $n \in \mathbb{N}$ and an *output type*, either "index" or "boolean". Function names $\mathtt{f}$ of arity n and type s range over n-ary functions from string indices to s.

Terms. Terms are given by the following grammar

$$\begin{aligned} T \rightarrow\ & \mathtt{x} \mid T_1 = T_2 \mid \top \mid \bot \mid \mathtt{f}(T_1, \ldots, T_k) \mid \\ & s(T_1) \mid p(T_1) \mid \sigma(T_1)\ (\sigma \in \Sigma_{\bowtie}) \mid \text{if } T_1 \text{ then } T_2 \text{ else } T_3 \end{aligned}$$

Terms inherit "boolean" or "index" types inductively from their variables and function names, and term formation is subject to the usual typing rules: for $\mathtt{f}(T_1, \ldots, T_k)$, $\sigma(T_1)$, $s(T_1)$ or $p(T_1)$, the type of each T_I must be "index"; for $T_1 = T_2$, the types of T_1 and T_2 must be the same; for and "if T_1 then T_2 else T_3," then the type of T_1 must be "boolean," and the types of T_2 and T_3 must agree.

Programs. A *program* consists of a tuple $(\mathtt{f}_1, \ldots, \mathtt{f}_k)$ of function names, plus k lines of the form $\mathtt{f}_i(\mathtt{x}_{1_i}, \ldots, \mathtt{x}_{n_i}) = T_i$, where T_i is a term whose type agrees with the output type of $\mathtt{f}_i$, every variable that occurs in T_i is some $\mathtt{x}_{\mathtt{i_j}}$, and every function name that occurs in T_i is some $\mathtt{f}_j$. Syntactically, we will write

$$\begin{aligned} \mathtt{f}_1(\vec{\mathtt{x}}_1) &= T_1(\vec{\mathtt{f}}, \vec{\mathtt{x}}_1) \\ &\vdots \\ \mathtt{f}_k(\vec{\mathtt{x}}_k) &= T_k(\vec{\mathtt{f}}, \vec{\mathtt{x}}_k) \end{aligned}$$

to indicate that the above properties hold.

Semantics. We impose the usual least fixed-point semantics on recursive programs. Briefly; over a given string, terms denote functionals which are monotone relative to extension relation on partial functions. We define the semantics of a program to be the first coordinate $\bar{f}_1$ of the least fixed-point $(\bar{f}_1, \ldots, \bar{f}_k)$ of the monotone operator $(f, \ldots, f_k) \mapsto (T_1(\vec{f}), \ldots, T_k(\vec{f}))$ [17].

Definition 3. *A* boolean monadic recursive scheme (BMRS) *is a program in which the arity of every function name in the program is one, and the output type of every function name in the program is "boolean."*

Boolean monadic recursive schemes compute (partial) functions from string indices to booleans, or equivalently (partial) subsets of indices. For example, the following scheme detects exactly those indices with some preceding b.

$$\mathtt{f}(\mathtt{x}) = \text{if } \rtimes (p(\mathtt{x})) \text{ then } \bot \text{ else if } b(p(\mathtt{x})) \text{ then } \top \text{ else } \mathtt{f}(p(\mathtt{x})) \tag{1}$$

4.2 Schemes as Definitions of String Transductions

We can define a string transduction $t : \Sigma^* \to \Gamma^*$ via a BMRS interpretation as follows. Fix a *copy set* $C = \{1, \ldots, m\}$ and for $n = |\Gamma|$ consider a system T of equations with a set of recursive functions $\vec{\mathtt{f}} = (\gamma_1^1, \ldots, \gamma_1^m, \gamma_2^1, \ldots, \gamma_n^m, \mathtt{f}_1, \ldots, \mathtt{f}_k)$; that is, with a function γ^c for each $\gamma \in \Gamma$ and $c \in C$.

Following the definition of logical string transductions [9,10], the semantics of T given an input model $\mathbf{S}$ with a universe D as follows. For each $d \in D$, we output a copy d^c of d if and only if there is exactly one $\gamma \in \Gamma$ for $c \in C$ such that $\gamma^c(\mathtt{x}) \in T$ evaluates to $\top$ when $\mathtt{x}$ is mapped to d. We fix the order of these output copies to be derived from C and the order on D induced by the predecessor function p: for any two copies d^c and d^e of a single index d, $d^c < d^e$ iff $c < e$ in the order on C, and for any copies d_i^c and d_j^e for distinct input indices d_i, d_j, $d_i^c < d_j^e$ iff $d_i < d_j$ in the order on the indices in $\mathbf{S}$. We fix the order due to the relation between order-preserving logical transductions and one-tape finite-state transducers [10].

This semantics of T thus defines a string transduction $t = t(T)$ where for a string $w \in \Sigma^*$ of length ℓ, $t(w) = u_0 u_1 ... u_\ell u_{\ell+1}$, where each $u_i = \gamma_1 ... \gamma_r$ if and only if for each γ_j, $1 \leq j \leq r$, γ_j is the unique symbol in Γ for $j \in C$ such that $\gamma_j^j(\mathtt{x})$ evaluates to $\top$ when $\mathtt{x}$ is assigned to i in the structure of $\rtimes w \ltimes$. An example is given in Example 2.

To describe partial functions we can add to $\vec{\mathtt{f}}$ a special function $\mathtt{def}(\mathtt{x})$ and specify the semantics of t to state that $t(w)$ is defined iff $\mathtt{def}(\mathtt{x})$ evaluates to $\top$ for element ℓ in w.

Example 2. The following is a BMRS definition of f from Example 1 using strings models from $\rtimes \Sigma^* \ltimes$. The copy set is $C = \{1, 2\}$.

$$\begin{array}{ll}
a^1(\mathtt{x}) = \text{if } a(\mathtt{x}) \text{ then} & c^1(\mathtt{x}) = \text{if } \ltimes(\mathtt{x}) \text{ then } b^1(p(\mathtt{x})) \text{ else } \bot \\
\qquad\quad \text{if } \rtimes(p(\mathtt{x})) \text{ then } \top \text{ else } b^1(p(\mathtt{x})) & c^2(\mathtt{x}) = \bot \\
\qquad\quad \text{else } \bot & d^1(\mathtt{x}) = \text{if } \ltimes(\mathtt{x}) \text{ then } a^1(p(\mathtt{x})) \text{ else } \bot \\
a^2(\mathtt{x}) = \bot & d^2(\mathtt{x}) = \bot \\
b^1(\mathtt{x}) = \text{if } a(\mathtt{x}) \text{ then } a^1(p(x)) \text{ else } \bot & \\
b^2(\mathtt{x}) = \text{if } a(\mathtt{x}) \text{ then } a^1(p(x)) \text{ else } \bot &
\end{array}$$

The following shows how this maps *aaaaa* to *abbabbad*:

	0	1	2	3	4	5	6
Input:	$\rtimes$	a	a	a	a	a	$\ltimes$
Copy 1:		a	b	a	b	a	d
Copy 2:			b		b		

We define two important variants of BMRS logic. For BMRS systems of equations over a set of recursive function symbols $\mathtt{f}$, we say a system of equations $T \in \text{BMRS}^p$ iff it contains no terms of the form $s(T_1)$ for any term T_1, and likewise $T \in \text{BMRS}^s$ iff it contains no terms of the form $p(T_1)$ for any term T_1. We define these as they fix the 'direction' of the recursion, which will be important in connecting them to the left- and right-subsequential functions.

4.3 Convergence and Well-Definedness

We only want to consider BMRS that compute well-defined transductions. Therefore, we require that for each string $w \in \Sigma^*$, each index i of w, and each $c \in C$ and $\gamma \in \Gamma$, every function $\gamma^c(i)$ converges, and furthermore for each c, there is a unique γ such that $\gamma^c(i) = \top$.

This is of course a semantic property, which is not an issue as far as the following proofs of extensional equivalence are concerned. However, there is an effective way of transforming a BMRS T into a BMRS T' such that T' computes a well-defined transduction, and agrees with T on inputs where T is well-defined.[2] Therefore, considering partially-defined schemata do not increase the computational power in any appreciable way.

5 Equivalence

5.1 Subsequential Functions Are BMRS-Definable

For a left-subsequential function $f : \Sigma^* \to \Gamma^*$, we can define an equivalent function in BMRS^p over models of strings in $\rtimes\Sigma^*\ltimes$. We do this by giving a construction of its SFST.

For an SFST $\mathcal{T} = \langle Q, q_0, Q_f, \delta, o, \omega\rangle$, where Q is the set of k states, we construct a BMRS^p system of equations T over the set of recursive functions $\vec{\mathtt{f}} = (\gamma_1^1, ..., \gamma_1^m, \gamma_2^1, ..., \gamma_n^m, \mathtt{q}_0, \ldots, \mathtt{q}_{k-1})$, where $n = |\Gamma|$ and m is the maximum length of any output of o or ω. The definitions in T are fixed as follows. First, we define $\mathtt{q}_0, \ldots, \mathtt{q}_{k-1}$ to parallel the transition function δ. For each state $q \in Q$ we define its corresponding recursive function symbol $\mathtt{q}$ as

$$\begin{aligned} \mathtt{q}(\mathtt{x}) = \quad & \quad\text{if } \mathtt{q}_1(p(\mathtt{x})) \text{ then } \sigma_1(\mathtt{x}) \\ & \text{else } \text{if } \mathtt{q}_2(p(\mathtt{x})) \text{ then } \sigma_2(\mathtt{x}) \\ & \text{else } ... \\ & \text{else } \text{if } \mathtt{q}_\ell(p(\mathtt{x})) \text{ then } \sigma_\ell(\mathtt{x}) \\ & \text{else } \bot \end{aligned} \tag{2}$$

where $q_1, ..., q_\ell$ is the set of states reaching q; that is, the set of states such that for each q_i, $\delta(q_i, \sigma_i) = q$. For the start state we instead set the final 'else' statement to $\mathtt{x}$ is the minimum element in the string; i.e. that $\rtimes(p(\mathtt{x}))$.

We then define the set of functions $\gamma_1^1, ..., \gamma_1^m, \gamma_2^1, ..., \gamma_n^m$ representing the symbols in the output strings to parallel the output and final functions o and ω:

$$\begin{aligned} \gamma^c(\mathtt{x}) = \quad & \quad\text{if } \mathtt{q}_1(\mathtt{x}) \text{ then } \sigma_1(\mathtt{x}) \\ & \text{else } \text{if } \mathtt{q}_2(\mathtt{x}) \text{ then } \sigma_2(\mathtt{x}) \\ & \text{else } ... \\ & \text{else } \text{if } \mathtt{q}_\ell(\mathtt{x}) \text{ then } \sigma_n(\mathtt{x}) \text{ else } \bot \end{aligned} \tag{3}$$

[2] For example, we can augment a boolean monadic recursive scheme with a "clock" that returns some default value if the program does not terminate within a given polynomial number of steps. (For each BMRS, there is some polynomial which bounds the number of steps in each terminating computation). Using a large "switch statement," we can ensure that exactly one character gets printed.

for all states q_i whose output on σ_i has γ as the cth symbol. That is, for each q_i either $o(q_i, \sigma_i) = u_1\gamma u_2$ or, if $\sigma_i = \ltimes$, that $\omega(q_i) = u_1\gamma u_2$, where $|u_1| = c - 1$. If there are no such states we set $\gamma^c(\mathtt{x}) = \bot$.

Finally, in cases when $Q_f \subsetneq Q$ we can, via the definition of the semantics of BMRS transductions for partial functions, we set the equation for the special function $\mathtt{def}(\mathtt{x})$ determining when the function is defined as

$$\begin{aligned} \mathtt{def}(\mathtt{x}) = \quad & \quad \text{if } \mathtt{q}_1(\mathtt{x}) \text{ then } \top \\ & \text{else} \quad \text{if } \mathtt{q}_2(\mathtt{x}) \text{ then } \top \\ & \text{else} \quad ... \\ & \text{else} \quad \text{if } \mathtt{q}_\ell(\mathtt{x}) \text{ then } \top \text{ else } \bot \end{aligned} \tag{4}$$

for $q_i \in Q_f$. When $Q_f = Q$, we set $\mathtt{def}(\mathtt{x}) = \top$.

An example definition modeling the SFST in Fig. 1, and an example computation for an input string $aaaa$ is given in Table 1.

Table 1. A BMRS transduction for the SFST in Fig. 1 (left) and an example derivation (right). The rows for $\mathtt{a}^2(\mathtt{x})$, $\mathtt{c}^2(\mathtt{x})$, and $\mathtt{d}^2(\mathtt{x})$ have been omitted.

$$\begin{aligned} \mathtt{def}(\mathtt{x}) &= \top \\ \mathtt{q}_0(\mathtt{x}) &= \text{if } \mathtt{q}_1(p(\mathtt{x})) \text{ then } a(\mathtt{x}) \text{ else } \rtimes(p(\mathtt{x})) \\ \mathtt{q}_1(\mathtt{x}) &= \text{if } \mathtt{q}_0(p(\mathtt{x})) \text{ then } a(\mathtt{x}) \text{ else } \bot \\ \mathtt{a}^1(\mathtt{x}) &= \text{if } \mathtt{q}_0(\mathtt{x}) \text{ then } a(\mathtt{x}) \text{ else } \bot \\ \mathtt{a}^2(\mathtt{x}) &= \bot \\ \mathtt{b}^1(\mathtt{x}) &= \text{if } \mathtt{q}_1(\mathtt{x}) \text{ then } a(\mathtt{x}) \text{ else } \bot \\ \mathtt{b}^2(\mathtt{x}) &= \text{if } \mathtt{q}_1(\mathtt{x}) \text{ then } a(\mathtt{x}) \text{ else } \bot \\ \mathtt{c}^1(\mathtt{x}) &= \text{if } \mathtt{q}_0(\mathtt{x}) \text{ then } \ltimes(\mathtt{x}) \text{ else } \bot \\ \mathtt{c}^2(\mathtt{x}) &= \bot \\ \mathtt{d}^1(\mathtt{x}) &= \text{if } \mathtt{q}_1(\mathtt{x}) \text{ then } \ltimes(\mathtt{x}) \text{ else } \bot \\ \mathtt{d}^2(\mathtt{x}) &= \bot \end{aligned}$$

Input:	$\rtimes$	a	a	a	a	$\ltimes$
	0	1	2	3	4	5
$\mathtt{q}_0(\mathtt{x})$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$
$\mathtt{q}_1(\mathtt{x})$	$\bot$	$\bot$	$\top$	$\bot$	$\top$	$\bot$
$\mathtt{a}^1(\mathtt{x})$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\bot$
$\mathtt{b}^1(\mathtt{x})$	$\bot$	$\bot$	$\top$	$\bot$	$\top$	$\bot$
$\mathtt{b}^2(\mathtt{x})$	$\bot$	$\bot$	$\top$	$\bot$	$\top$	$\bot$
$\mathtt{c}^1(\mathtt{x})$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$
$\mathtt{d}^1(\mathtt{x})$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
Output:						
Copy 1 :		a	b	a	b	c
Copy 2 :			b		b	

Lemma 1. *Any left-subsequential function has some BMRS*^{p} *definition.*

Proof. It is sufficient to show that the above construction creates from any SFST $\mathcal{T}$ a BMRS^p system of equations T whose transduction $t(T) = t(\mathcal{T})$.

Consider any string in $w = \sigma_1...\sigma_n \in \Sigma^*$ of length n; we refer to the positions in $\rtimes w \ltimes$ as their indices $0, 1, ..., n+1$. From the construction $\mathtt{q}_0(\mathtt{x})$ is always true of position 1; likewise by definition $\mathcal{T}$ is in state q_0 at position 1. By definition (2) for T, whenever $\mathcal{T}$ is in state q_i, reads position i, and $\delta(q_i, \sigma_i) = q_j$, then $\mathtt{q_j}(\mathtt{x})$ in T evaluates to $\top$ for $i+1$, because 'if $\mathtt{q_i}(\mathtt{x})$ then $\sigma_i(\mathtt{x})$' is in the definition for $\mathtt{q_j}(\mathtt{x})$. By induction on δ^* it is thus the case that whenever $\mathcal{T}$ is in state q_i at position i, position i satisfies $\mathtt{q_i}(\mathtt{x})$ in T.

Let $o(q_i, \sigma_i) = u = \gamma_1 ... \gamma_m$ for any position i in w. By (3), for each γ_j there is a function $\gamma_j^j(\mathtt{x})$ whose definition includes 'if $\mathtt{q_i}(\mathtt{x})$ then $\sigma_i(x)$'. Because i satisfies $\mathtt{q_i}(\mathtt{x})$ in T, then each jth copy of i will be γ_j, and so the output of i under T will also be $\gamma_1 ... \gamma_j = u$. This also holds for the output function ω.

Thus for any w, $t(\mathcal{T})(w) = w'$ implies that $t(T)(w) = w'$, and it is not hard to show that the reverse holds. □

The following lemma shows that the same is true for right-subsequential functions and BMRS^s, which follows by the same logic as for Lemma 1.

Lemma 2. *Any right-subsequential function has some BMRS^s definition.*

5.2 BMRS^p and BMRS^s-Definable String Functions Are Subsequential

To show the converse, we show that for any well-defined BMRS^p transduction T, for $f = t(T)$, the sets $\{f_x \mid x \in \Sigma^*\}$ are finite. For a copy set $C = \{1, ..., m\}$ and for $n = |\Gamma|$ consider a system T of equations with a set of recursive functions

$$\vec{\mathtt{f}} = (\gamma_1^1, ..., \gamma_1^m, \gamma_2^1, ..., \gamma_n^m, \mathtt{f}_1, \ldots, \mathtt{f}_k);$$

let F be the set of function names appearing in $\vec{\mathtt{f}}$, and let ℓ be the maximum number such that $\mathtt{x} - \ell$ appears as a term in T.

First, define $\mathtt{sats}(w, i) = \{\mathtt{f} \in T \mid \mathtt{f}(i) = \top \text{ in } w\}$ to identify the functions in T true in w at index i. The following fact will be used throughout this proof.

Remark 1. For any $\mathtt{f} \in F$ and string $w \in \rtimes\Sigma^*\ltimes$, the value of $\mathtt{f}(i)$ can be calculated from the sets $F_\ell, F_{\ell-1}, ..., F_1$, where for each $1 \leq j \leq \ell$, $F_j = \mathtt{sats}(w, i-j)$.

Proof. Let $\mathtt{f}(\mathtt{x}) = T_\mathtt{f}$ be the equation for $\mathtt{f}$ in T. By the definition of T, ℓ is the maximum number of times the p function can be applied to a variable in any term in T. Thus, for any function $\mathtt{g} \in F$, $T_\mathtt{f}$ can only contain $\mathtt{g}(\mathtt{x} - h)$ for at most some $h \leq \ell$. Thus, in terms of the semantics of $\mathtt{f}(i)$ for some index i in w, the value of $\mathtt{g}(i - h)$ can be determined by whether $\mathtt{g}$ is in F_h. The remainder of the semantics of $\mathtt{f}(i)$ then follows from the definition of the semantics of BMRSs. □

The following states that $\mathtt{sats}(w, i)$ holds no matter how w is extended. This follows directly from Remark 1.

Remark 2. For any $w, v \in \rtimes\Sigma^*\ltimes$, $\mathtt{sats}(w, i) = \mathtt{sats}(wv, i)$.

Recall that among the functions in F there is a function $\gamma^c \in F$ for each $\gamma \in \Gamma$ and $c \in C$. Recall also that the semantics of BMRS transductions produces an output string u_i at each input index i such that γ is the cth position in u_i if and only if $\gamma^c(i)$ evaluates to $\top$. (The stipulation that there is only one such γ^c ensures that only a single output string is produced for each index). To refer to this string we define $\mathtt{out}(w, i) = \gamma_1\gamma_2...\gamma_h$ where each $\gamma_j \in \mathtt{sats}(w, i)$.

Then let $\texttt{out}_{\texttt{T}}(w) = \texttt{out}(w,1)\cdot\texttt{out}(w,2)\cdot\ldots\cdot\texttt{out}(w,\texttt{last}(w))$, where $\texttt{last}(w)$ indicates the final index in w.

We can now connect these facts to the string function $f = t(T)$ described by T. Recall the technicality that the domain of f is Σ^* but T is defined over string models of the form $\rtimes\Sigma^*\ltimes$. First, the above allows us to make the following assertion about the relationship between $\texttt{out}_T$ and f^p.

Remark 3. $\texttt{out}_{\texttt{T}}(\rtimes w) \sqsubseteq f^p(w)$.

Proof. This follows directly from Remark 2: the output at each index at w will be constant no matter how w is extended. Thus, $f^p(w)$ at least includes $\texttt{out}_T(\rtimes w)$.

The final piece is to define when two strings w and v are equivalent with respect to T, which we then show that they are equivalent with respect to f; that is, that $f_w = f_v$. Intuitively, w and v are equivalent if their final ℓ indices satisfy exactly the same functions in F. Formally,

$$w \equiv_T v \text{ iff for all } 0 \leq i < \ell, \texttt{sats}(\rtimes w, \texttt{last}(\rtimes w) - i) = \texttt{sats}(\rtimes v, \texttt{last}(\rtimes v) - i)$$

Remark 4. The partition on Σ^* induced by $\equiv_T$ is finite.

Proof. For any sequence of ℓ indices, there are at most $(2^{|F|})^\ell$ possible sequences of subsets of F that they satisfy.

The following states the key implication that equivalence with respect to T implies equivalence with respect to f.

Lemma 3. *For any two strings $w, v \in \Sigma^*$, $w \equiv_T v$ implies $f_w = f_v$.*

Proof. First, for any $\sigma \in \Sigma_{\ltimes}$, $\texttt{out}(\rtimes w\sigma, \texttt{last}(\rtimes w\sigma)) = \texttt{out}(\rtimes v\sigma, \texttt{last}(\rtimes v\sigma))$. In other words, the string output at any additional σ following w and v is the same. This follows from Remark 1 and the fact that the final ℓ indices in $\rtimes w$ and $\rtimes v$ satisfy the same sets of functions in F.

For any string $u \in \Sigma^*$, then, by induction on the length of $u\ltimes$ it is clear that $f(wu) = \texttt{out}_T(\rtimes w)u'$ and $f(vu) = \texttt{out}_T(\rtimes v)u'$ for the same $u' \in \Gamma^*$. From this and Remark 3, we know that $f^p(w) = \texttt{out}_T(\rtimes w)u_1$ and $f^p(v) = \texttt{out}_T(\rtimes v)u_1$ for the same $u_1 \in \Gamma^*$. Clearly then for any $u \in \Sigma^*$, $f(wu) = f^p(w)u'$ and $f(vu) = f^p(v)u'$ and so by the definition of f_w and f_v, $f_w = f_v$.

Lemma 4. *For any BMRS^p transduction T, the function $f = t(T)$ is a left-subsequential function.*

Proof. The set $\{f_x \mid x \in \Sigma^*\}$ is finite: from Remark 4, $\equiv_T$ induces a finite partition on Σ^*, and by Lemma 3, for any two strings w, v in the same block in this partition, $f_w = f_v$. Thus there can only be finitely many such functions f_x.

We omit the proof for the following parallel lemma for BMRS^s.

Lemma 5. *For any BMRS^s transduction T, the function $f = t(T)$ is a right-subsequential function.*

5.3 Main Theorem

We now can give the central result of the paper.

Theorem 3. *BMRS^p (respectively, BMRS^s) transductions are equivalent to the left-subsequential (resp., right-subsequential) functions.*

Proof. From Lemmas 1, 2, 4, and 5.

6 Discussion

The above result provides the first logical characterization of the subsequential functions. A consequence of this is we can get a better understanding of subclasses of the subsequential functions. We sketch two here.

First, the *input strictly local* (ISL) functions are a strict subset of the subsequential class for which the output string is computed by referencing a bounded window in the *input* string only [2,3]. Briefly, a function is ISL iff there is some number k such that for any two strings w and v that share a $k-1$ suffix, $f_w = f_v$. This class has attractive learnability properties [3] and empirically is relevant to processes in natural language phonology [5]. We omit a proof, but it is almost certainly the case that a BMRS system of equations T corresponds to an ISL function iff for each function symbol $\mathtt{f} \in \vec{\mathtt{f}}$, the definition of $\mathtt{f}$ contains no recursive function calls. This is further interesting in that it suggests that any left-subsequential function f has a ISL counterpart whose input alphabet subsumes the recursive function symbols in the BMRS^p definition of f.[3] This is strongly reminiscent of the old result that any regular language is the homomorphism of a strictly 2-local language [15].

A sister class to the ISL functions is the *output strictly local* (OSL) functions, which are those subsequential functions which compute the output string by referencing the current input and a bounded window in the output [2,4]. They are divided into two classes the left- and right-OSL functions depending on whether the string is read from the left or right. We conjecture that a BMRS system of equations T corresponds to an OSL function iff for each function $\mathtt{f_i} \in \mathtt{f}$ corresponding to $\gamma^c \in \Gamma$, for any non recursively-defined $\sigma(t)$ $(\sigma \in \Sigma)$, then $t = x$. BMRS^p systems of equations of this type correspond to left-OSL functions, while BMRS^s systems of this type correspond to right-OSL functions.

Finally, this paper has limited its discussion to BMRS transductions restricted to either p or s, so an obvious open question is to what functions are described by BMRS transductions without this restriction. As any rational function is the composition of a right- and left-subsequential function [8], it is clear that BMRS transductions in general are strictly more expressive than either the BMRS^p and BMRS^s transductions. Based on this, we tentatively conjecture that the BMRS transductions in general are equivalent to the rational functions, but this claim requires more rigorous investigation than can be done here.

[3] We thank Jeff Heinz for pointing this out.

7 Conclusion

This paper has given the first logical characterization of the subsequential functions. As with previous work connecting logical, language-theoretic, and automata-theoretic characterizations of formal languages and functions, we are confident this will further study of the connections between subclasses of the subsequential functions, and subclasses of the rational functions in general.

References

1. Büchi, J.R.: Weak second-order arithmetic and finite automata. Z. Math. Log. Grundl. Mathmatik **6**, 66–92 (1960)
2. Chandlee, J.: Strictly Local Phonological Processes. Ph.D. thesis, University of Delaware (2014)
3. Chandlee, J., Eyraud, R., Heinz, J.: Learning strictly local subsequential functions. Trans. Assoc. Comput. Linguist. **2**, 491–503 (2014)
4. Chandlee, J., Eyraud, R., Heinz, J.: Output strictly local functions. In: Kornai, A., Kuhlmann, M. (eds.) Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2014), Chicago, IL, pp. 52–63, July 2015
5. Chandlee, J., Heinz, J.: Strictly locality and phonological maps. Linguist. Inq. **49**, 23–60 (2018)
6. Chandlee, J., Jardine, A.: Autosegmental input-strictly local functions. Trans. Assoc. Comput. Linguist. **7**, 157–168 (2019)
7. Elgot, C.C.: Decision problems of finite automata design and related arithmetics. Trans. Am. Math. Soc. **98**(1), 21–51 (1961)
8. Elgot, C.C., Mezei, J.E.: On relations defined by generalized finite automata. IBM J. Res. Dev. **9**, 47–68 (1965)
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. Filiot, E.: Logic-automata connections for transformations. In: Banerjee, M., Krishna, S.N. (eds.) ICLA 2015. LNCS, vol. 8923, pp. 30–57. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-45824-2_3
11. Filiot, E., Reynier, P.: Transducers, logic, and algebra for functions of finite words. ACM SIGLOG News **3**(3), 4–19 (2016)
12. Heinz, J.: The computational nature of phonological generalizations. In: Hyman, L., Plank, F. (eds.) Phonological Typology. Phonetics and Phonology, pp. 126–195. De Gruyter Mouton, Berlin (2018). Chapter 5
13. Heinz, J., Lai, R.: Vowel harmony and subsequentiality. In: Kornai, A., Kuhlmann, M. (eds.) Proceedings of the 13th Meeting on Mathematics of Language, Sofia, Bulgaria, pp. 52–63 (2013)
14. McNaughton, R., Papert, S.: Counter-Free Automata. MIT Press, Cambridge (1971)
15. Medvedev, Y.T.: On the class of events representable in a finite automaton. In: Moore, E.F. (ed.) Sequential Machines - Selected Papers, pp. 215–227. Addison-Wesley, New York (1964)
16. Mohri, M.: Finite-state transducers in language and speech processing. Comput. Linguist. **23**(2), 269–311 (1997)
17. Moschovakis, Y.N.: Abstract Recursion and Intrinsic Complexity. Lecture Notes in Logic, vol. 48. Cambridge University Press, Cambridge (2019)

18. Oncina, J., García, P., Vidal, E.: Learning subsequential transducers for pattern recognition tasks. IEEE Trans. Pattern Anal. Mach. Intell. **15**, 448–458 (1993)
19. Schützenberger, M.P.: Sur une variante des fonctions séquentielles. Theor. Comput. Sci. **4**, 47–57 (1977)
20. Thomas, W.: Classifying regular events in symbolic logic. J. Comput. Syst. Sci. **25**, 360–376 (1982)
21. Trakhtenbrot, B.A.: Finite automata and logic of monadic predicates. Dokl. Akad. Nauk SSSR **140**, 326–329 (1961)