A logical characterization of strictly local functions

Jane Chandlee and Steven Lindell, Haverford College

We characterize the input strictly local functions of Chandlee by quantifier-free logical interpretations over strings with adjacency.

§ 1 Introduction

It has been argued that the sub-regular hierarchy provides a more appropriate setting to study phonological processes than the full power of finite state transducers [1, 2]. Although restricted finite state machines can provide an intuitive way to model these cognitive processes, it is argued there that logical definability is a more foundational way to understand these mechanisms, independent of their implementation.

From a language recognition point of view, the strictly local languages (and more recently strictly piecewise languages) are at the bottom of that hierarchy [3, 4]. However, recognizing a valid string in a language is only one piece of the puzzle. In generative phonology, what is truly desired is a description of the input-output functions that map an underlying representation to its surface representation. It has been argued that the vast majority of such mappings can be characterized as strictly local functions [5]. These are essentially functions computed by local automata – those that can only keep track of the previous k symbols in memory [6]. In particular, these local automata appear to be the simplest machines that can produce nontrivial functions of interest to linguists.

In this paper, evidence of this simplicity is presented by showing that these strictly local functions correspond to quantifier-free interpretations in mathematical logic. In the process of proving this correspondence, we discover a natural condition on strictly local functions that guarantees their composability. This condition can be simply stated as mappings which preserve infinitude – the image of an infinite set must be infinite.

The rest of the paper is organized as follows. In section 2 we provide the relevant background and definitions required for strictly local functions and quantifier free interpretations. In section 3 we prove their equivalence, and section 4 provides directions for further work.

§ 2 Preliminaries

Before we connect local transductions with quantifier free interpretations, we need some preliminary definitions in the areas of language theory and model theory.

Language theory

We refer the reader to [6] for the basics about finite-state transducers.

For any finite alphabet Σ of symbols, Σ^* is the free monoid generating finite strings over Σ. For any non-empty set of strings S, let $\wedge S$ be the *longest common prefix* of S.

Definition [6, p. 692]: For a function $f : \Sigma^* \to \Gamma^*$ and $x \in \Sigma^*$ let $f_x(y) = (\wedge f[x\Sigma^*])^{-1} f(xy)$ be the *tail* of f at x. Note that $\wedge f[x\Sigma^*]$ is always a prefix of $f(xy)$ for any y, so the use of inverse makes sense here.

It is well-known that f can be computed by a finite state transducer iff its set of tails $\{f_x : x \in \Sigma^*\}$ is finite. And if its output depends only on the most recent k inputs, we say that the function is *k-local*.

Definition [5]: A function f is *input strictly local* (ISL) if for some k and all u, $f_{ux} = f_x$ for all $x \in \Sigma^{k-1}$.

The following is sometimes taken as the definition of a local function, but is also proved in [5].

Fact: A function from Σ^* to Γ^* is k-ISL if and only if it can be computed by a Markovian Mealy automaton whose states correspond to $\Sigma^{<k}$, input strings of length less than k. In a Mealy machine each transition between states is labeled with an input symbol from Σ and an output string from Γ^*.

In order to ensure closure under composition among local functions, we need to bound the ratio between their output and input lengths. The easiest way to enforce this would be to forbid the automaton from having a null cycle – a closed circuit of transitions all of which output the empty string. However, there is a more elegant language theoretic way of expressing this.

Definition: A function $f : \Sigma^* \to \Gamma^*$ is *non-degenerate* if for all infinite $L \subseteq \Sigma^*$, $f[L]$ is also infinite.

Claim: Suppose a function f is computed by minimal finite state machine M. Then f is non-degenerate if and only if M has no null cycles.

Proof: Suppose M has a null cycle C labeled by v. Since every state in M is reachable, we can get to C from the start state via some string u, and then $f[uv^*] = f[u]$, which makes f degenerate. On the other hand, if M has n states, every path of length n contains a cycle, so every sequence of n symbols must produce some output. So a string w must produce an output of length at least $|w| \div n$. If L is infinite, it contains arbitrarily long strings, thereby producing arbitrarily long output, and so $f[L]$ is infinite.

Model Theory

We represent strings as a particular kind of structure with unary functions and predicate symbols.

Definition: A *singulary* structure is a structure of the form $\langle D; f, \ldots, R, \ldots\rangle$ where D is a set (the domain), each $f : D \to D$ is a unary function, and each $R \subseteq D$ is a unary predicate.

Example: We use a particular type of singulary structure to represent strings. For example, *baab* will be represented as a $\langle\{1, 2, 3, 4\}; s, p, U_a, U_b\rangle$ where s is a successor function, p is a predecessor function, and $\{U_a, U_b\}$ partition the domain $\{1, 2, 3, 4\}$ into the locations of the symbols a and b. I.e.

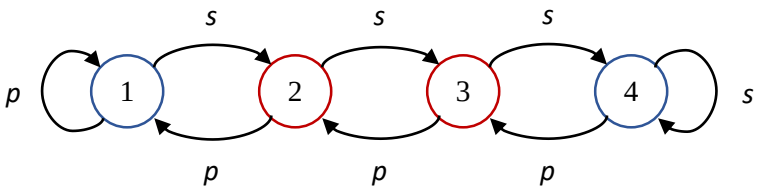

Figure 1, the structure for the string *baab*, where red (a) is $U_a = \{2, 3\}$ and blue (b) is $U_b = \{1, 4\}$.

Note that s and p are inverses of each other, except when they reach their maximum and minimum respectively. In the sequel we will notate them +1 and −1 for convenience, even though this will lead to some peculiar looking situations at the ends such as 4 + 1 = 4 and 1 – 1 = 1 as shown in the figure above. This is easily generalized to arbitrary finite strings over an arbitrary finite alphabet.

Definition: A Σ-*structure* is a tuple $\langle \{1, \ldots, n\}; s, p, \{U_\sigma : \sigma \in \Sigma\}\rangle$ where $s(i) = i + 1$ for $i < n$ with $s(n) = n$, and $p(i) = i - 1$ for $i > 1$ with $p(1) = 1$, such that $\{U_\sigma : \sigma \in \Sigma\}$ is a partition of the domain. Notice that this corresponds to a unique string in Σ^* of length n whose i^{th} symbol is σ when $U_\sigma(i)$ is true.

We assume the reader is familiar with the basics of first-order interpretations on structures. Just as finite state automata provide a mechanism for string to string transductions, logical formulae provide a way to define mappings from one structure to another, known as translations or interpretations. Although the general notion of an interpretation permits defining a mapping from any logical signature to another (i.e. a translation), we confine ourselves to mappings from strings to strings. In particular, we restrict ourselves to linear interpretations that produce an output consisting of m copies of the input for some fixed m, defined by vectors of formulas. We use superscript notation to denote various copies of formulas, variables, elements, or sets. E.g. d^c represents the c^{th} copy of an element d, and is really just an abbreviation for $\langle d, c\rangle \in D \times C$ for some copy set $C = \{1, \ldots, m\}$. In addition, for clarity we make use of ±1 for predecessor and successor, and polymorphic terms to define polymorphic functions by cases instead of their more complicated graphs. Let polymorphic formulas $\varphi^C(x) = \langle \varphi^1(x), \ldots, \varphi^m(x)\rangle$ be vectors of ordinary formulas parametrized by variable type, so that $\varphi^C(x^c) = \varphi^c(x)$ for copies $c \in C$. For two variables we write $\theta^{C^2}(x^c, y^{c'}) = \theta^{c,c'}(x, y)$. But we drop the superscript on polymorphic formulas whenever their free variables are typed. For example, $\varphi(x^c) = \varphi^C(x^c)$ because we assume that if $\varphi(x^c)$ makes sense, then it must be $\varphi^C(x^c)$ for some C containing c. In other words, a formula is polymorphic if and only if its variables are typed.

Example: Look at an interpretation to double every symbol (e.g. *baab* → *bbaaaabb*), with $m = 2$. Define two copies x^1 and x^2 of every input element, placing them in the adjacency relation below. Informally, we define the successor (+ 1) and predecessor (− 1) functions by cases on the left. On the right are the polymorphic formulas $\sigma^{C^2}(x, y)$ and $\pi^{C^2}(x, y)$ specifying those definitions, $C = \{1, 2\}$.

Definition	conditions	interpretation
$x^1 + 1 = x^2$	always	$\sigma^{12}(x, y) \equiv [y = x]$
$x^2 + 1 = (x + 1)^1$	if $x + 1 \neq x$	$\sigma^{21}(x, y) \equiv [y = s(x) \neq x]$
$= x^2$	if $x + 1 = x$	$\sigma^{22}(x, y) \equiv [y = x = s(x)]$
$x^2 - 1 = x^1$	always	$\pi^{21}(x, y) \equiv [y = x]$
$x^1 - 1 = (x - 1)^2$	if $x - 1 \neq x$	$\pi^{12}(x, y) \equiv [y = p(x) \neq x]$
$= x^1$	if $x - 1 = x$	$\pi^{11}(x, y) \equiv [y = x = p(x)]$

Use $\upsilon_a(x^1) \equiv \upsilon_a(x^2) \equiv U_a(x)$ and $\upsilon_b(x^1) \equiv \upsilon_b(x^2) \equiv U_b(x)$ to say the copies are replicas of the originals.

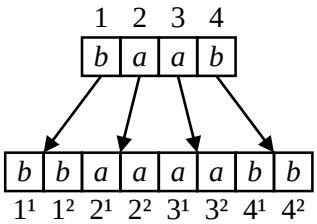

Figure 2, illustrating an instance of the 2-copy interpretation, *baab* → *bbaaaabb*.

The generalized version is defined as follows.

Definition: An *m-copy interpretation* from Σ-structures to Γ-structures is a tuple of polymorphic formulas $\langle \varphi^C(x); \pi^{C^2}(x, y), \sigma^{C^2}(x, y), \{\upsilon_\gamma{}^C(x) : \gamma \in \Gamma\}\rangle$ over copies $C = \{1, \ldots, m\}$ which assign to each Σ-structure A a Γ-structure B whose output domain $|B| = \{a^c : A \vDash \varphi^c[a], a \in |A|, c \in C\}$ is drawn from m disjoint copies of the input domain $|A| \times C$. Adjacency between these elements is determined by formulas $\pi^{c,c'}(x, y)$ and $\sigma^{c,c'}(x, y)$ which collectively are the graphs of polymorphic functions on $|B|$, representing the predecessor $x^c - 1 = y^{c'}$ and successor $x^c + 1 = y^{c'}$ respectively. The formulas $\{\upsilon_\gamma{}^c(x) : c \in C, \gamma \in \Gamma\}$ determine a polymorphic partition of $|B|$ whose pieces $\{a^c \in |B| : A \vDash \upsilon_\gamma{}^c[a]$ for $c = 1, \ldots, m\}$ are the places where the symbol γ appears.

Because the domain formula φ above does not involve multiple variables (only copies), we call these *singulary* interpretations. But we need to be careful about quantifier free polymorphic function graphs. For even without copies, the definition $\psi(x, y) \equiv [y = y - 1]$ is the graph of a function that maps everything to the bottom element, intuitively violating a sense of locality, and only computable with a kind of search. So instead, as in the example above, we will require the more intuitive definition by cases notation to describe π and σ. E.g. in an m-copy interpretation in which all m copies are present, the following defines a valid successor function (generalizing the example above to arbitrary m):

$x^c + 1$	$= x^{c+1}$	if $1 \le c < m$; else	{the next copy if it is not the last}
	$= (x + 1)^1$	if $x + 1 \ne x$ (and $c = m$); else	{the first copy of the next element}
	$= x^c$	otherwise ($c = m$ and $x + 1 = x$)	{the last copy of the top element}

In general, a polymorphic function $f : |A| \times C \to |A| \times C$ defined by cases using first-order terms and conditions can be converted into a polymorphic formula $\psi^{C^2}(x, y)$ defining the graph of f via $A \vDash \psi^{c,c'}[a, f(a)] \Leftrightarrow f(a^c) \in |A|^{c'}$ for all $c, c' \in C$. However, as we have just observed, the reverse is not true because even a quantifier-free formula can implicitly define the graph of a function that will not have an explicit definition by cases.

§ 3 Results

An interpretation is monotone if copies of an element maintain the same order as their originals. We write $x \le y$ to mean x precedes y (or equivalently y succeeds x).

Definition: An interpretation is *monotone* if $x \le y$ only if $x^c \le y^{c'}$ for any copies c and c'. This is easily seen to be equivalent to the statement that every copy of x occurs before/after every copy of $x \pm 1$. This ensures that all copies of a given element remain together in blocks.

Here is the main result of this paper.

Theorem: A function $f : \Sigma^* \to \Gamma^*$ is local and non-degenerate if and only if it can be described by a monotone quantifier-free singulary interpretation in which the functions are defined by cases.

Proof: Let M be a k^{th}-order Markovian automaton without null cycles which computes f, whose states are $\Sigma^{<k}$. For each state $q = \sigma_1 \ldots \sigma_i \in \Sigma^{<k} = Q$, where $1 \le |q| = i < k$, let $\delta : Q \times \Sigma \to \Gamma^*$ be the output transition function. That is, $\delta(q, \sigma) \in \Gamma^*$ is the string that is output when the past k symbols (fewer if we are close to the beginning of the input) have been $q\sigma$.

The idea is to create the output domain by *interleaving* copies of the input domain. The number of copies required is given by the maximum length output string for any single input symbol. I.e. set $m = \max\{|\delta(q, \sigma)| : q \in Q, \sigma \in \Sigma\}$. But the construction is complicated by the fact that the number of output symbols per input symbol varies over the course of the computation.

In the following, we design formulas to describe what state we are in, based on the position in the input string, as represented by the variable x. We can monitor how close we are to the beginning by:

$$\begin{array}{lcll} \psi_1(x) & \equiv & x - 1 = x & \text{(at the first position)} \\ \psi_j(x) & \equiv & x - j = x - (j-1) \neq x - (j-2) & \text{(at position } j\text{, for } 1 < j < k\text{)} \\ \psi_k(x) & \equiv & x - (k-1) \neq x - (k-2) & \text{(at position } k \text{ or higher)} \end{array}$$

The formula $\theta_q(x) \equiv U_{\sigma_1}(x - i) \wedge \ldots \wedge U_{\sigma_i}(x - 1) \wedge \psi_{i+1}(x)$ says the machine is in state q at place x of the input. Now it is possible to ascertain exactly how many copies of each input element are required by looking at the length of the output (anywhere from 0 to m) at that point in the automaton. Let $\varphi^C(x)$ define the output domain as:

$$\varphi^c(x) \equiv \bigvee\{\theta_q(x) \wedge U_\sigma(x) : q \in Q, \sigma \in \Sigma, c \leq |\delta(q, \sigma)|\} \qquad \text{for } 1 \leq c \leq m$$

That is to say, there will be one copy for each symbol in the output string at point x. Note that there will be no copies if the output string is empty at that point. We interleave these in such a way as to keep all outputs of a given input element x consecutive in copy order x^1, …, ensuring monotonicity. Here is the definition by cases for just the predecessor function, because the successor is similar.

$$\begin{array}{lll} x^c - 1 = x^{c-1} & \text{for } c > 1\text{; else for some } 1 \leq c' \leq m\text{:} \\ \quad (x-1)^{c'} & \text{if } x - 1 \neq x \wedge \varphi^{c'}(x-1) \wedge \neg\varphi^{c'+1}(x-1) \text{ else} \\ \quad \ldots & \\ \quad (x - |Q|)^{c'} & \text{if } x - |Q| \neq x - (|Q| - 1) \wedge \varphi^{c'}(x - |Q|) \wedge \neg\varphi^{c'+1}(x - |Q|) \\ \quad x^1 & \text{otherwise} \end{array}$$

In other words, the predecessor of an output element is the preceding extant copy of that element, or else the last copy of the previous extant element. Failing those, it must be the first element. N.b. that non-degeneracy guarantees some output has been generated in the prior $|Q|$ steps, unless we are close to the beginning of the input, so that either way the number of cases is always bounded.

Finally, we can define which symbols appear at each position of the output. Let

$$\upsilon_\gamma(x^c) \equiv \bigvee\{\theta_q(x) \wedge U_\sigma(x) : q \in Q, \sigma \in \Sigma, \text{whenever } \gamma \text{ is the } c^{\text{th}} \text{ symbol of } \delta(q, \sigma)\}$$

which means that $\gamma \in \Gamma$ is placed on copy x^c of the output string.

In the other direction, let $\langle \varphi^C(x);\ \pi^{C^2}(x, y),\ \sigma^{C^2}(x, y),\ \{\upsilon_\gamma{}^C(x) : \gamma \in \Gamma\}\rangle$ be an m-copy interpretation mapping Σ-structures to Γ-structures. As before, φ defines the domain copies. For clarity of exposition, we will ignore the successor functions s and σ in the input and output structures, and work only with the predecessor functions p and π, both written as -1. This overloading of notation will be clear from context as $x - 1 = p(x)$ for untyped x that refer to the input, and $\pi(x^c, x^c - 1)$ for typed variables that refer to the output. Remember that π is a monotone function on those domain copies, forcing them to appear in blocks, but some blocks may not exist and in others the copies may be out of order 1, …, m (there is no reason they have to be). But we can figure that all out by subtracting one until we change blocks (or hit the bottom). For $1 \leq j \leq m$, the polymorphic

$$\beta_j(x^c) \equiv x^c - (j-1) = x^{c'} \wedge x^c - j = (x - i)^{c'} \vee x^c - 1 = x^c \qquad \text{(taking a disjunction over } c' \in C\text{)}$$

is a formula which says that the c^{th} copy of x is in the j^{th} position of its output block. It does this by observing that it takes j applications of predecessor to change blocks to $x - i$. Notice that $x - i$ is simply some term of π in its definition by cases, where $i > 0$.

Now let $(x - k)$ be the smallest predecessor term occurring in any formula of the interpretation (i.e. k is the largest number of consecutive applications of p). Let M be a $(k+1)$-order Markov machine with states $Q = \Sigma^{\leq k}$ and transitions between them assigned in the usual manner.

Since any such formula $\psi(x)$ depends only upon x and at most k of its predecessors, we can determine its truth value at any transition $\delta(\sigma_i \ldots \sigma_1, \sigma_0)$ of M by taking $q = \sigma_i \ldots \sigma_1 \in Q$ and $\sigma_0 \in \Sigma$ to derive a formula $\psi[\sigma_i \ldots \sigma_1 \sigma_0]$ meaning $\psi(x)$ is true in state q when reading symbol σ_0, independent of x. Construct $\psi[\sigma_i \ldots \sigma_1 \sigma_0]$ by replacing each atomic relation by a Boolean value. Since there are no constants and only one function, every term must be of the form $x - j$ for $j = 0, \ldots k$. The only relations are the symbol predicates U_σ and of course equality, =. So $U_\sigma(x - j)$ becomes true if $\sigma = \sigma_{\min(j,\, i)}$ and false otherwise (because the predecessor bottoms out by definition). And $x - j_1 = x - j_2$ becomes true if $\min(j_1, i) = \min(j_2, i)$ and false otherwise (for the same reason).

To define the outputs generated by each transition of the Markov process, assign the j^{th} symbol of $\delta(\sigma_i \ldots \sigma_1, \sigma_0)$ to be γ whenever $\upsilon_\gamma{}^c[\sigma_i \ldots \sigma_0]$ and $\beta_j{}^c[\sigma_i \ldots \sigma_0]$. The resulting machine M computes the transduction. To see it has no null cycles, observe that if it did, there would be arbitrarily long portions of the input that produce no output, which implies that the predecessor function π would have to jump an arbitrary amount backwards, which is impossible because that limit is k.

§ 4 Conclusion

We have seen that there is a machine independent way of characterizing those local transductions on strings which are closed under composition, corresponding to non-degenerate input strictly local functions. That characterization uses quantifier-free formulas in logic to monotonically translate between structures representing strings with adjacency. It is hoped that a similar logical characterization can be obtained for an analogous notion of strictly piecewise functions [4].

References

[1] Rogers, J., Heinz, J., Fero, M., Hurst, J., Lambert, D., and Wibel, S. (2013). Cognitive and sub-regular complexity. In Morrill, G. and Nederhof, M.-J., editors, *Formal Grammar, Lecture Notes in Computer Science*, volume 8036, pages 90-108. Springer.

[2] Rogers, J. and Pullum, G. (2011). Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information*, vol. 20:329-342.

[3] Jäger, G. and Rogers, J. (2012). Formal language theory: Refining the Chomsky hierarchy, Philosophical Transactions of the Royal Society B, volume 367:1956–1970.

[4] Rogers, J., Heinz, J., Bailey, G., Edlefsen, M., Visscher, M., Wellcome, D., and Wibel, S. On languages piecewise testable in the strict sense. In Christian Ebert, Gerhard Jäger, and Jens Michaelis, editors, *The Mathematics of Language*, volume 6149 of Lecture Notes in Artifical Intelligence, pages 255–265. Springer, 2010.

[5] Chandlee, J. (2014). Strictly Local Phonological Processes, Ph.D. thesis, University of Delaware.

[6] Sakarovitch J. (2009). *Elements of automata theory*, Cambridge University Press.