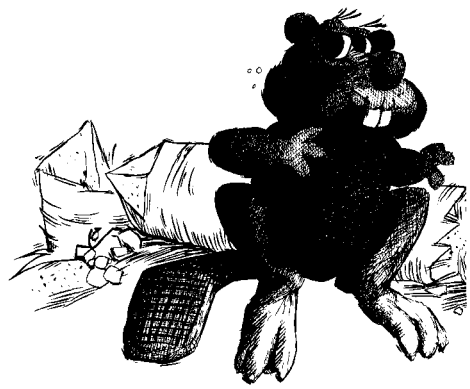
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT



*Single-Valued Finite Transduction**

J. Howard Johnson

CS-86-59

November, 1986

Single-Valued Finite Transduction*

J. Howard Johnson
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

November 6, 1986

Abstract

Finite state transduction is a simple and effective tool for the efficient analysis and transformation of large bodies of text. However, it is sometimes difficult or undesirable to specify single-valued transductions directly. Cases where multiple output values are generated can then be handled in one of the following ways: (1) Consider the input to be in error. (2) Select the shortest word and consider input in error if ties remain. (3) Select the genealogical minimum of the possible outputs (minimum length with lexicographic minimum in case of ties). (4) Select the lexicographic minimum of the possible outputs. (5) Resolve ambiguous union operations by selecting the first alternative. These strategies are discussed in terms of a generic one-pass algorithm which scans its input from left to right and writes output symbols as they can be determined. The strategies are implemented by providing appropriate *-semirings as the implementation of an abstract data type used in the generic algorithm. This approach suggests a number of practical algorithms in addition to unifying the study of these models. For example, it leads to an algorithm for constructing minimum state subsequential transducers satisfying the minimum delay property as well as providing a new view of rational and subsequential functions.

1 Introduction

The lexical analysis of natural and artificial languages involves a great deal of character-by-character processing which must be done efficiently. On the other hand, the nature of the processing can be quite complex so that a high level specification language would facilitate use, especially in an environment

*This work was supported by the Natural Sciences and Engineering Research Council of Canada, Grant No. A0237.

where *ad hoc* requests need to be handled. Thus a model is needed which (1) allows the specification of the transformations in a natural readable way and (2) yields efficient implementations possibly after optimization.

Single-valued finite state transduction [AU72,Ber79] is one such model. It allows the specification of many of the transformations occurring in practice as rational expressions [Ber79,Joh83]. The closure of finite transduction under composition means that complex models can be made up of simpler models cascaded together [EM65,Ber79]. They also subsume many of the models based on regular sets which are commonly used in the lexical analysis of programming languages or text editors [AU72,LS78,ASU86]. Although they are incapable of counting or matching parentheses, they make up for this defect by allowing the resolution of ambiguity through unbounded lookahead. In this they behave like the class of LR-Regular languages [CC73] but without some of the undecidability problems. Furthermore, in cases where only bounded lookahead is required a particularly efficient implementation (deterministic GSM mappings or subsequential transduction) [Gin66,Sch77,Ber79] is available.

There is, however, another problem which comes up in the specification of transductions. Rational expressions describe finite transductions which are not necessarily single-valued but since we usually want our transformation to yield a single output for a given input, we have to make the transduction single-valued or guarantee that this is not a problem. Of course, single-valuedness of finite transductions has been known to be decidable for some time [Sch75,BH77], but there are, as we will see, other solutions possible which are slightly more general and yet efficiently implementable.

In principle, inputs which cause multiple output values to be generated can be handled in one of the following ways:

(1) Consider the input to be in error.

(2) Select the shortest output. Consider input in error if there is more than one output of the same length.

(3) Select the genealogical minimum of the possible outputs (minimum length with lexicographic minimum in case of ties).

(4) Select the lexicographic minimum of the possible outputs if defined.

(5) Whenever the multiple valuedness is caused by a union of two cases and the first alternative is single-valued, take this first alternative. Consider the input in error if this does not make the output unique. This modified union operation will be called an "elseor".

Note that (1), (2), and (5) cause the domain of the transduction to be redefined to be that part which defined but does not cause an error. The

domain can also be restricted in case (4) since the lexicographic minimum is not always defined. Clearly, the power of these models now exceeds conventional single-valued finite transduction, that is, finite transduction which are always single-valued by themselves.

Of course, one would not like to be forced to compute the set of possible outputs and then perform the selection, but would rather combine the selection with the transduction process itself. In many cases this can be done, resulting in huge savings. The technique for this is described as a generic one-pass algorithm which scans its input from left to right and writes output symbols as soon as they can be determined. The strategies are implemented by providing appropriate $*$-semirings as implementation for an abstract data type used in the generic algorithm.

This approach suggests a number of practical algorithms in addition to unifying the study of these models. For example, it leads to an algorithm for constructing minimum state subsequential transducers satisfying the minimum delay property as well as providing a new view of rational and subsequential functions.

Single-valued finite transductions have been used successfully in the computerization of the Oxford English Dictionary, in particular, for the tag enhancement and analysis of the raw text. It also promises to be a key component in a query processing facility to allow efficient *ad hoc* access.

This paper is organized as follows: Section 2 provides the necessary background of $*$-semirings to fix notation and provide a basis for further discussion. Section 3 describes a model for $*$-semiring automata again mainly to fix notation. Section 4 outlines the generic one-pass algorithm which is used to achieve all of the above procedures. Section 5 introduces the single-value $*$-semiring which is used to implement strategy (1) in conjunction with the method of section 4. Sections 6, 7, and 8 address strategies (2), (3) and (4) by defining appropriate $*$-semirings. Section 9 explains how the technique of section 8 can be modified to implement strategy (5). Section 10 briefly addresses the side issue of how a truly on-line algorithm can be obtained for traditional single-valued transduction and why it does not generalize to the new models. Section 11 discusses deterministic transduction and how it can be obtained from the models of section 4. Section 12 applies the techniques of sections 10 and 11 together with the single-value $*$-semiring to yield an algorithm for computing optimal subsequential transducers.

2 $*$-Semirings

An *$*$-semiring* is a system $\langle S, \oplus, \odot, ^*, \mathbf{0}, \mathbf{1}\rangle$ where S is a set closed under the binary operations $\oplus$ (addition) and $\odot$ (multiplication), the unary operation * (asteration), and there are elements $\mathbf{0}$ and $\mathbf{1}$ such that the following laws

are satisfied for all $a, b, c \in S$:

(1)	$a \oplus (b \oplus c) = (a \oplus b) \oplus c,$	$\oplus$ is associative
(2)	$a \oplus b = b \oplus a,$	$\oplus$ is commutative
(3)	$a \oplus \mathbf{0} = a,$	$\mathbf{0}$ is an identity for $\oplus$
(4)	$a \odot (b \odot c) = (a \odot b) \odot c,$	$\odot$ is associative
(5)	$a \odot \mathbf{1} = \mathbf{1} \odot a = a,$	$\mathbf{1}$ is a identity for $\odot$
(6)	$a \odot \mathbf{0} = \mathbf{0} \odot a = \mathbf{0},$	$\mathbf{0}$ is a zero for $\odot$
(7)	$a \odot (b \oplus c) = (a \odot b) \oplus (a \odot c),$	$\odot$ is left distributive over $\oplus$
(8)	$(a \oplus b) \odot c = (a \odot c) \oplus (b \odot c),$	$\odot$ is right distributive over $\oplus$
(9)	$a^* = (a \odot (a^*)) \oplus \mathbf{1} = ((a^*) \odot a) \oplus \mathbf{1}.$	* is an asterate for $\oplus$ and $\odot$

An *idempotent $*$-semiring* is one which, in addition, satisfies:

(10)	$a \oplus a = a$	$\oplus$ is idempotent

With slight variations $*$-semirings or idempotent $*$-semirings have been discussed by a number of authors [AHU74,BC75,Leh77,SS78,AS85,KS86].

Multiplication will usually be indicated by juxtaposition and will be assumed to have a higher priority that addition. Asteration is assumed to have higher priority than multiplication. If no confusion will result $\oplus$ will be written as $+$. With these conventions law (9) becomes $a^* = aa^* + \mathbf{1} = a^*a + \mathbf{1}$.

Example 2.1 *The Boolean $*$-semiring $\mathcal{B} = \langle \{0,1\}, +, \cdot, ^*, 0, 1 \rangle$ is an idempotent $*$-semiring where*

$$0 + 0 = 0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0$$

$$0 + 1 = 1 + 0 = 1 + 1 = 1 \cdot 1 = 0^* = 1^* = 1.$$

Example 2.2 *The class $\langle \mathbf{Reg}(\Sigma^*), \cup, \cdot, ^*, \emptyset, \{\varepsilon\} \rangle$ of regular languages over an alphabet Σ is an idempotent $*$-semiring.*

Example 2.3 *The set $\mathcal{N}_\infty$ of non-negative integers extended with ∞ is a non-idempotent $*$-semiring where for all $a \geq 1$*

$$a + \infty = \infty + a = \infty + \infty = 0 + \infty = \infty + 0 = \infty$$

$$a\infty = \infty a = \infty\infty = a^* = \infty^* = \infty$$

$$0\infty = \infty 0 = 0 \qquad 0^* = 1.$$

Example 2.4 *The set of m by m matrices whose elements are from a $*$-semiring K is again a $*$-semiring: $\langle K^{m \times m}, \oplus, \odot, ^*, \mathbf{0}, \mathbf{I} \rangle$. Here $\oplus$ and $\odot$ applied to matrices are the usual definitions of matrix addition and multiplication where the base elements use the operations from K. The matrix*

$\mathbf{0}$ is an m by m matrix containing only the value $\mathbf{0}$. The matrix $\mathbf{I}$ is an m by m matrix containing the value $\mathbf{1}$ on the principal diagonal and $\mathbf{0}$ elsewhere. The asterate $X = A^*$ of a matrix A is the unique solution to the system $AX \oplus \mathbf{I} = X$. The value of X may be found using a modified Gaussian elimination or Gauss-Jordan reduction or using the idea of eliminants. Algorithms for the finding the solution to the above system as well as its uniqueness are discussed in [BC75,Leh77,Car79,AS85]. Note that $K^{m\times m}$ will be idempotent if K is.

A $*$-semiring morphism is a mapping between $*$-semirings which preserves all of the operations. Thus if S_1 and S_2 are $*$-semirings and f is a mapping from S_1 to S_2 then f must satisfy, for all $a, b, c \in S_1$:

$$f(a \oplus b) = f(a) \oplus f(b) \qquad f(a \odot b) = f(a) \odot f(b)$$

$$f(a^*) = f(a)^* \qquad f(\mathbf{0}) = \mathbf{0} \qquad f(\mathbf{1}) = \mathbf{1}$$

Note that S_2 must be idempotent if S_1 is. Any expression of the form $f(E)$ where E involves only elements and operations from S_1 can be then replaced by an expression E' where each element from S_1 has been replaced by its image in S_2 and each operator from S_1 has been replaced by the corresponding operator in S_2. If the structure of S_2 is simpler than that of S_1 this can yield a saving in computation cost.

Example 2.5 *The mapping from* $\mathbf{Reg}(\Sigma^*)$ *to* $\mathcal{B}$ *which maps* $\emptyset$ *to* 0 *and anything else to* 1 *is a morphism.*

Example 2.6 *Morphisms from* $\mathbf{Reg}(\Sigma^*)$ *to* $\mathbf{Reg}(\Delta^*)$ *can be specified by identifying the image for each letter from* Σ. *These mappings can be used to encode one alphabet into another, to alias letters from* Σ, *or to erase letters from* Σ.

Example 2.7 *There can be no $*$-semiring morphism from* $\mathbf{Reg}(\Sigma^*)$ *to* $\mathcal{N}_\infty$ *since the first of these is idempotent and the second is not.*

Since we have a notion of multiplication, we can introduce a notion of divisibility. We will say that u left divides v (written $u \mid v$) if there is a w such that $u \odot w = v$. If this value is unique then we will call w the left quotient of u into v and write $w = u \backslash v$. Left divisibility satisfies the reflexive property $(u \mid u)$ and the transitive property $(u \mid v, v \mid w \Longrightarrow u \mid w)$. Any elements u and v satisfying $u \mid v$ and $v \mid u$ are called *associates*. This defines an equivalence relation of associates. If, on the other hand, there are no distinct associates then $\mid$ satisfies the anti-symmetric law $(u \mid v, v \mid u \Longrightarrow u = v)$ and $\mid$ defines a partial order.

A common (left) divisor for two elements u and v is any element w such that $w \mid u$ and $w \mid v$. Note that $\mathbf{1}$ is always a common divisor. A greatest common left divisor w of u and v satisfies:

$$w \mid u, \quad w \mid v, \qquad (z \mid u, z \mid v) \Longrightarrow z \mid w.$$

Thus the greatest common divisor is a common divisor which any other common divisor must divide. If it exists it will be unique up to associate class. We can then introduce the operation $w = u \wedge v$ which yields for any values u and v a distinguished associate, that is, one chosen to represent the class.

Whenever $\wedge$ is defined, it will satisfy the laws:

(1)	$a \wedge (b \wedge c) = (a \wedge b) \wedge c,$	$\wedge$ is associative
(2)	$a \wedge b = b \wedge a,$	$\wedge$ is commutative
(3)	$a \wedge a = a,$	$\wedge$ is idempotent
(4)	$a \wedge \mathbf{1} = \mathbf{1}.$	$\mathbf{1}$ is an zero for $\wedge$

Since $\wedge$ is commutative and associative, it makes sense to talk about the $\bigwedge$ of a set of elements. A $*$-semiring for which $\wedge$ is always defined will be called a *GCLD* $*$-semiring.

Example 2.8 *The class $\mathcal{N}_\infty$ has the usual notion of integer divisibility with the additional observation that any nonzero element divides ∞ and any element divides 0. The $\wedge$ of two numbers is their integer GCD and $a \wedge \infty = a$ for any nonzero a.*

In a number of cases the greatest common left divisor will not always exist or will be inappropriate for our purposes. In such cases we will identify a particular common left divisor for consideration. The designated common left divisor of u and v will be denoted $u \tilde{\wedge} v$. The only required property of this operation will be that $w = u \tilde{\wedge} v$ implies that $w \mid u$ and $w \mid v$ and that w is in some, perhaps weak, sense maximal for this property. Since this operation will be applied to a set of values, commutativity and associativity is desired. If these properties do not hold then the $\tilde{\bigwedge}$ of a set of values will depend on the order of evaluation.

3 $*$-Semiring Automata

A *K-Nondeterministic Finite Automaton* (K-NFA) $T = \langle \Sigma, K, \delta, s, F \rangle$ is a 5-tuple where Σ is a finite alphabet, K is a $*$-semiring, $\delta: \Sigma \cup \{\varepsilon\} \mapsto K^{m \times m}$ is a mapping from $\Sigma \cup \{\varepsilon\}$ to square matrices over K, $s \in K^{1 \times m}$ is a row vector of elements from K, and $F \in K^{m \times 1}$ is a column vector of elements from K.

It is then possible to define the closure of δ as $\hat{\delta}(x) = \delta(x)\delta^*(\varepsilon)$ for all $x \in \Sigma$. Here $\delta^*(\varepsilon)$ is the unique solution X to the system of equations $\delta(\varepsilon)X + \mathbf{I} = X$ and will usually be equivalent to the infinite sum

$$\mathbf{I} + \delta(\varepsilon) + \delta^2(\varepsilon) + \delta^3(\varepsilon) + \delta^4(\varepsilon) + \cdots.$$

The mapping $\hat{\delta}$ can be extended to Σ^* by defining $\hat{\delta}(ux) = \hat{\delta}(u)\hat{\delta}(x)$ for all $u \in \Sigma^*$ and $x \in \Sigma$ and $\hat{\delta}(\varepsilon) = \mathbf{I}$. The closure of s will be defined as $\hat{s} = s\delta^*(\varepsilon)$. The behaviour $|T|$ of T is then defined to be a mapping from Σ^* into K: $|T|(w) = \hat{s}\hat{\delta}(w)F$ for all $w \in \Sigma^*$. Thus T associates with each word in Σ^* a K value according to the above computation.

Note that for input $x_1x_2x_3\cdots x_n$ the computed value is exactly

$$s\delta^*(\varepsilon)\delta(x_1)\delta^*(\varepsilon)\delta(x_2)\delta^*(\varepsilon)\delta(x_3)\delta^*(\varepsilon)\cdots\delta(x_n)\delta^*(\varepsilon)F.$$

Because of the implied associativity of matrix multiplication there are several ways in which this result may be computed.

Theorem 3.1 *Let $T = \langle \Sigma, K_1, \delta, s, F\rangle$ be a K_1-NFA and f be a $*$-semiring morphism from K_1 to K_2. Furthermore, let δ' be a mapping from $\Sigma^* \cup \{\varepsilon\}$ to K_2 satisfying $\delta'(x)_{ij} = f(\delta(x)_{ij})$ and let $s'_i = f(s_i)$ and $F'_i = f(F_i)$ for all $i, j \leq m$. Finally, let $T' = \langle \Sigma, K_2, \delta', s', F'\rangle$. Then for every $w \in \Sigma^*$, $|T'|(w) = f(|T|(w))$.*

Proof: For any $w \in \Sigma^*$ the definition of T yields an expression E in terms of elements of the $*$-semiring K_1. The expression E can be transformed to an expression in K_2 equal to $f(E)$ by replacing each K_1 element by its image in K_2 and each operation in K_1 by the corresponding operation in K_2. □

Note that when $\delta(\varepsilon) = \mathbf{0}$ we have that $\delta^*(\varepsilon) = \mathbf{I}$. The above expression then simplifies to:

$$s\delta(x_1)\delta(x_2)\delta(x_3)\cdots\delta(x_n)F.$$

Example 3.1 *If K is the Boolean $*$-semiring, then the definition of $\mathcal{B}$-Nondeterministic Finite Automaton is exactly the standard Nondeterministic Finite Automaton as described in [HU79,Har78]. The state set is the set of m index values. The vectors s and F identify a set of start states and final states, respectively. There is a path from state i to state j not beginning with ε and with label u exactly when $\hat{\delta}(u)_{ij} = 1$. The result $|T|(w)$ will equal 1 exactly when there is a path from a start state to a final state with label w.*

A *K-Deterministic Finite Automaton* (K-DFA) $T = \langle \Sigma, K, \delta, s, F\rangle$ is a K-NFA satisfying the additional constraints:

1. s has only one nonzero value,

2. $\delta(x)$ has only one nonzero value in each row for all $x \in \Sigma$, and

3. $\delta(\varepsilon) = 0$.

Example 3.2 *If K is the Boolean $*$-semiring, then the definition corresponds to the standard notion of Deterministic Finite Automaton. The condition on s implies a single start state and the condition on δ implies that no state has more than one outgoing transition with the same label. The condition of $\delta(\varepsilon)$ requires the absence of ε transitions.*

Example 3.3 *Let K be $\mathcal{N}_\infty$. The above models correspond to automata which count the number of ways a word belongs to a set. This is often called the* ambiguity *of a word and corresponds to the number of distinct parses according to some grammar.*

Example 3.4 *Let K be* $\mathbf{Reg}(\Sigma^*)$. *Then these models correspond to different models of finite state transduction (not necessarily single-valued). This is closely related to the matrix representation of a finite transduction [Ber79].*

4 Left to Right Recognition

Suppose that we are given a K-NFA T and an input word and wish to compute $|T|(x_1x_2x_3\cdots x_n)$. This can be done by forming the product from left to right as in

$$\left(\cdots\left(\left(\left(\hat{s}\hat{\delta}(x_1)\right)\hat{\delta}(x_2)\right)\hat{\delta}(x_3)\right)\cdots\hat{\delta}(x_n)F\right).$$

If the routine *read_character* obtains the input one character at a time, yielding $\dashv$ when the input is exhausted, this may be represented by the following algorithm:

Algorithm 4.1

$x : \Sigma$; $v : K^{1\times m}$
$v := \hat{s}$;
$x :=$ *read_character*;
while $x \neq \dashv$ **do begin**
    $v := v\hat{\delta}(x)$;
    $x :=$ *read_character*
end;
write(vF);

Note that $v\hat{\delta}(x)$ may be computed in a number of different ways depending on the properties of the $*$-semirings concerned. These details, though interesting in themselves, are out of scope for this paper.

If K is a GCLD $*$-semiring, then the following observation can be used to yield a modified algorithm that often has better performance without sacrificing correctness. Suppose that at some point in the execution there is a value z which left divides all of the components of v. Since one of the components of v must left divide the result value and left division is a transitive relation, we can immediately write out z and replace v by the result of the left division of v by the scalar value z:

Algorithm 4.2

$x : \Sigma$; $z : K$; $v : K^{1\times m}$
$v := \hat{s}$;
while true do begin
  $z := \bigwedge v$;
  write(z);
  $v := z \setminus v$;
  $x :=$ *read_character*;
  if $x = \dashv$ **then exit loop**;
  $v := v\hat{\delta}(x)$
end;
write(vF);

Here $\bigwedge v$ is the value $v_1 \wedge v_2 \wedge v_3 \wedge \cdots \wedge v_m$ and $z \setminus v$ is the vector formed by left dividing each element of v by z. If K is not a GCLD $*$-semiring then $\wedge$ may be replaced by $\tilde{\wedge}$ as discussed in section 2.

The approach taken in the next five sections will involve the definition of a number of $*$-semirings which are each the image of a $*$-semiring morphism applied to $\mathbf{Reg}(\Delta^*)$. Then, given a morphism from $\mathbf{Reg}(\Delta^*)$ to K, theorem 3.1 will allow any $\mathbf{Reg}(\Delta^*)$-NFA to be be transformed to a K-NFA which will behave like the original machine followed by the morphism. Then algorithm 4.2 can be used to perform the computations in K, thereby saving time since K has a simpler structure.

5 Censoring Multiple-Valued Outputs

The first approach to making a transduction single-valued is to simply throw away the multiple-valued part, that is, report a domain error in cases where an input value can produce more than one output. This will be achieved using the Single-Value $*$-semiring $\mathbf{SV}(\Delta^*)$ together with theorem 3.1 and algorithm 4.2.

The algebra $\mathbf{SV}(\Delta^*) = \langle \Delta^* \cup \{0, \top\}, \oplus, \odot, ^*, 0, \varepsilon \rangle$ is formed by adding to the free monoid Δ^* the two elements

0: indicating the absence of a value, and

$\top$: indicating a multiplicity of values.

The multiplicative operation $\odot$ is that of the free monoid for elements from Δ^*. Multiplication of elements from $\Delta^* \cup \{\top\}$ on either side by $\top$ yields $\top$ and multiplication of elements from $\Delta^* \cup \{0, \top\}$ on either side by 0 yields 0. It is easily shown that $\odot$ is associative, has ε as a identity (left and right), and 0 as a zero (left and right).

The additive operation $\oplus$ is idempotent so that any value added to itself yields the same value. Addition of distinct values from Δ^* yields $\top$ as does anything added to $\top$ on either side. The element 0 is a left and right identity. It is easily shown that $\oplus$ is associative, commutative, idempotent, and has 0 as an identity.

The star of 0 and ε is ε. The star of other values is $\top$. The star property $((x^* \odot x) \oplus \varepsilon = (x \odot x^*) \oplus \varepsilon = x^*)$ is easily verified.

The following tables summarize $\oplus$, $\odot$, and *. Here u and v are arbitrary distinct elements from Δ^+.

$\oplus$	0	ε	u	v	$\top$
0	0	ε	u	v	$\top$
ε	ε	ε	$\top$	$\top$	$\top$
u	u	$\top$	u	$\top$	$\top$
v	v	$\top$	$\top$	v	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$	$\top$

$\odot$	0	ε	u	v	$\top$
0	0	0	0	0	0
ε	0	ε	u	v	$\top$
u	0	u	uu	uv	$\top$
v	0	v	vu	vv	$\top$
$\top$	0	$\top$	$\top$	$\top$	$\top$

*	
0	ε
ε	ε
u	$\top$
v	$\top$
$\top$	$\top$

Lemma 5.1 *The distributive laws* $u \odot (v \oplus w) = (u \odot v) \oplus (u \odot w)$ *and* $(v \oplus w) \odot u = (v \odot u) \oplus (w \odot u)$ *hold for any* $u, v, w \in \mathbf{SV}(\Delta^*)$.

Proof: The left distributive law is easily established by case analysis:

$v = w$ or $w = 0$: Both sides reduce to $u \odot v$.

$v = 0$: Both sides reduce to $u \odot w$.

$u = 0$: Both sides equal 0.

$u \neq 0, v \oplus w = \top$: The left side is $\top$. If v or w is $\top$ the right side is $\top$. If $v, w \in \Delta^*$, $v \neq w$ the right side is $uv \oplus uw = \top$.

The right distributive law follows by an analogous argument. □

Theorem 5.2 $\mathbf{SV}(\Delta^*)$ *is an idempotent* $*$*-semiring.*

Proof: The above discussion establishes that the additive structure of $\mathbf{SV}(\Delta^*)$ is an idempotent commutative monoid, the multiplicative structure is a monoid, the additive identity is a multiplicative zero, and the star property holds. The distributive laws hold by Lemma 5.1. □

Let φ be the following mapping from $\mathbf{Reg}(\Delta^*)$ to $\mathbf{SV}(\Delta^*)$:

$$\varphi(L) = \begin{cases} 0 & \text{if } L = \emptyset \\ u & \text{if } L = \{u\} \\ \top & \text{if } |L| \geq 2 \end{cases} \qquad \text{for all } L \in \mathbf{Reg}(\Delta^*).$$

Theorem 5.3 *φ is a $*$-semiring morphism.*

Proof: It is necessary to show that this mapping preserves the operations of a $*$-semiring:

$\varphi(\emptyset) = 0$: Given.

$\varphi(\{\varepsilon\}) = \varepsilon$: Immediate from definition.

$\varphi(u \cup v) = \varphi(u) \oplus \varphi(v)$: Both $\cup$ and $\oplus$ are idempotent. The result follows immediately if $u = v$ or one of u and v is the empty set. Otherwise, if u and v are distinct singleton sets or one of u and v contains two or more elements, then both sides yield $\top$.

$\varphi(uv) = \varphi(u) \odot \varphi(v)$: The concatenation of singleton sets is a singleton set. The concatenation of a set with two or more elements with another nonempty set, on either side, is a set with two or more elements. The concatenation of the empty set with another set, on either side, is the empty set. In each case both sides agree.

$\varphi(u^*) = \varphi(u)^*$: Both sides yield ε if $u = \emptyset$ or $u = \varepsilon$. Otherwise both sides yield $\top$.

□

If both u and v are from Δ^* then $u \mid v$ iff u is a (not necessarily proper) prefix of v. Otherwise, any element in $\Delta^* \cup \{0, \top\}$ divides 0 and any element in $\Delta^* \cup \{\top\}$ divides $\top$.

If $u \mid v$ where $u, v \in \Delta^*$ then $u \setminus v$ equals the suffix of v which remains after u is removed. Otherwise, any element in $\Delta^* \cup \{\top\}$ divided into 0 yields 0 and any element in Δ^* divided into $\top$ yields $\top$. The cases $0 \setminus 0$ and $\top \setminus \top$ are multi-defined and will be conventionally assigned the values 0 and $\top$, respectively.

The following tables summarize $\mid$ and $\setminus$. Here u, v, and z are arbitrary elements from Δ^+ satisfying $v = uz$.

$\mid$	0	ε	u	v	$\top$
0	$\surd$				
ε	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
u	$\surd$		$\surd$	$\surd$	$\surd$
v	$\surd$			$\surd$	$\surd$
$\top$	$\surd$				$\surd$

$\backslash$	0	ε	u	v	$\top$
0	(0)				
ε	0	ε	u	v	$\top$
u	0		ε	z	$\top$
v	0			ε	$\top$
$\top$	0				($\top$)

We can define the $\wedge$ of elements of Δ^* as their greatest common prefix. The $\wedge$ of 0 or $\top$ with $\top$ is $\top$ and $0 \wedge 0 = 0$.

The following table summarizes the $\wedge$ operation. Here u, v, and z are arbitrary elements from Δ^+ satisfying $v = uz$.

$\wedge$	0	ε	u	v	$\top$
0	0	ε	u	v	$\top$
ε	ε	ε	ε	ε	ε
u	u	ε	u	u	u
v	v	ε	u	v	v
$\top$	$\top$	ε	u	v	$\top$

Lemma 5.4 *$\wedge$ defines a GCLD operation with respect to $\mid$.*

Proof: Let $u = v_1 \wedge v_2$ and consider the following cases:

$v_1, v_2 \in \Delta^*$: If z divides v_1 and v_2 it must be a common prefix and must thus divide u.

$v_1 \in \Delta^*, v_2 \in \{0, \top\}$: If z divides v_1 it must be a prefix of v_1.

$v_1 = \top, v_2 \in \{0, \top\}$: If z divides *top* it must be in $\Delta^* \cup \{\top\}$.

$v_1 = 0, v2 = 0$: Any z divides 0.

The other cases follow by commutativity of the definitions. □

The above process can be summarized by the following algorithm for computing $\bigwedge A$ where A is a possibly infinite subset of $\mathbf{SV}(\Delta^*)$:

> If A has at least one element from Δ^* then $\bigwedge A$ is the longest common prefix of the Δ^* elements. Otherwise if $\top \in A$ then $\bigwedge A = \top$. Otherwise $A = \{0\}$ and $\bigwedge A = 0$.

Theorem 5.5 $\mathbf{SV}(\Delta^*)$ *is a GCLD $*$-semiring. Furthermore $\bigwedge$ is defined even for infinite sets.*

Theorem 3.1 and algorithm 4.2 may then be used to transform any finite transducer to a $\mathbf{SV}(\Delta^*)$-NFA and apply it to an input. If a result from Δ^* is obtained then this is the output. Otherwise, a value of 0 or $\top$ is obtained and a domain error reported. The overall effect is to take the single-valued part of the transduction.

6 Selecting the Shortest Output

The second approach to making a transduction single-valued is to throw away all outputs which are longer than the shortest output. If this leaves a multiplicity of values, a domain error is reported. Otherwise the single output remaining is taken. This will be achieved using the Length-Minimum $*$-semiring $\mathbf{LM}(\Delta^*)$ together with theorem 3.1 and algorithm 4.2.

First of all we will define the *length partial order* $<$ on Δ^* as follows: $u < v$ if and only if $|u| < |v|$. Then $u \leq v$ if and only if $u < v$ or $u = v$.

The algebra $\mathbf{LM}(\Delta^*) = \langle \Delta^* \cup \{0\} \cup \{\top\}^+, \oplus, \odot, *, 0, \varepsilon \rangle$ will now be defined. The elements are those of Δ^* extended with 0 and non-empty strings of $\top$ elements.

The multiplicative operation $\odot$ is that of the free monoid for elements from Δ^*. Multiplication of any element by 0 on either side yields 0. Multiplication on either side of any element $u \in \Delta^*$ by a $\top^i$ yields $\top^{i+|u|}$. The product of $\top^i$ and $\top^j$ is $\top^{i+j}$. It is easily shown that $\odot$ is associative, has ε as a identity (left and right), and 0 as a zero (left and right).

The additive operation $\oplus$ applied to elements u and v of $\Delta^* \cup \{\top\}^+$ yields the shorter of the two if they have different lengths and $\top^{|u|}$ if they have the same length. The sum of $\top^i$ and $\top^j$ is $\top^{\min(i,j)}$. The element 0 is a left and right identity. It is easily shown that $\oplus$ is associative, commutative, idempotent, and has 0 as an identity.

The star of any element is ε. The star property $((x^* \odot x) \oplus \varepsilon = (x \odot x^*) \oplus \varepsilon = x^*)$ is easily verified.

The following tables summarize $\oplus$, $\odot$, and $\star$. Here u, v, and w are arbitrary distinct elements of Δ^+ where $|u| = |v| = i$ and $|u| < |w| = j$.

$\oplus$	0	ε	u	v	w	$\top^i$	$\top^j$
0	0	ε	u	v	w	$\top^i$	$\top^j$
ε	ε	ε	ε	ε	ε	ε	ε
u	u	ε	u	$\top^i$	u	$\top^i$	u
v	v	ε	$\top^i$	v	v	$\top^i$	v
w	w	ε	u	v	w	$\top^i$	$\top^j$
$\top^i$	$\top^i$	ε	$\top^i$	$\top^i$	$\top^i$	$\top^i$	$\top^i$
$\top^j$	$\top^j$	ε	u	v	$\top^j$	$\top^i$	$\top^j$

$\odot$	0	ε	u	v	w	$\top^i$	$\top^j$
0	0	0	0	0	0	0	0
ε	0	ε	u	v	w	$\top^i$	$\top^j$
u	0	u	uu	uv	uw	$\top^{2i}$	$\top^{i+j}$
v	0	v	vu	vv	vw	$\top^{2i}$	$\top^{i+j}$
w	0	w	wu	wv	ww	$\top^{i+j}$	$\top^{2j}$
$\top^i$	0	$\top^i$	$\top^{2i}$	$\top^{2i}$	$\top^{i+j}$	$\top^{2i}$	$\top^{i+j}$
$\top^j$	0	$\top^j$	$\top^{i+j}$	$\top^{i+j}$	$\top^{2j}$	$\top^{i+j}$	$\top^{2j}$

$*$	
0	ε
ε	ε
u	ε
v	ε
w	ε
$\top^i$	ε
$\top^j$	ε

Lemma 6.1 *The distributive laws* $u \odot (v \oplus w) = (u \odot v) \oplus (u \odot w)$ *and* $(v \oplus w) \odot u = (v \odot u) \oplus (w \odot u)$ *hold for any* $u, v, w \in \mathbf{LM}(\Delta^*)$.

Proof: The left distributive law is easily established by case analysis:

$v \leq w$ or $w = 0$: Both sides reduce to $u \odot v$.

$w \leq v$ or $v = 0$: Both sides reduce to $u \odot w$.

$u = 0$: Both sides equal 0.

$u \neq 0, v \oplus w = \top^{|v|}$: Both sides are $\top^{|u|+|v|}$.

The right distributive law follows by an analogous argument. □

Summing up these observations we have the following theorem.

Theorem 6.2 $\mathbf{LM}(\Delta^*)$ *is an idempotent* $*$*-semiring.*

Let ξ be the following mapping from $\mathbf{Reg}(\Delta^*)$ to $\mathbf{LM}(\Delta^*)$:

$$\xi(L) = \begin{cases} 0 & \text{if } L = \emptyset \\ u & \text{if } u \in L \text{ and } u \leq x \text{ for all } x \in L \\ \top^i & \text{if } \exists u, v \in L \text{ such that } |u| = |v| = i \text{ and } |x| \geq i \text{ for all } x \in L \end{cases}$$

for all $L \in \mathbf{Reg}(\Delta^*)$.

Theorem 6.3 ξ *is a* $*$*-semiring morphism.*

Proof: It is necessary to show that this mapping preserves the operations of a $*$-semiring:

$\xi(\emptyset) = 0$: Given.

$\xi(\{\varepsilon\}) = \varepsilon$: Immediate from definition.

$\xi(u \cup v) = \xi(u) \oplus \xi(v)$: Both $\cup$ and $\oplus$ are idempotent. The result follows immediately one of u or v is the empty set or contains a word shorter than any in the other set. Otherwise, if u and v have shortest words of the same length i so that both sides yield $\top^i$.

$\xi(uv) = \xi(u) \odot \xi(v)$: Since the shortest words in uv will be composed of the shortest words from u and v, consider only the shortest words from u and v. The concatenation of singleton sets is a singleton set. The concatenation of a set with two or more elements with another nonempty set, on either side, is a set with two or more elements. The concatenation of the empty set with another set, on either side, is the empty set. In each case both sides agree.

$\xi(u^*) = \xi(u)^*$: Both sides yield ε.

□

If both u and v are from Δ^* then $u \mid v$ iff u is a (not necessarily proper) prefix of v. Otherwise, any element divides 0 and any non-zero element shorter than i divides $\top^i$. Any $\top^i$ divides $\top^j$ if $i \leq j$.

If $u \mid v$ where $u, v \in \Delta^*$ then $u \setminus v$ equals the suffix of v which remains after u is removed. Otherwise, any non-zero element divided into 0 yields 0 and any element in $u \in \Delta^*$ such that $|u| < i$ divided into $\top^i$ yields $\top^{i-|u|}$. The case $\top^i \setminus \top^j$ is undefined when $i \geq j$. The cases $0 \setminus 0$ and $\top^i \setminus \top^j$ when $i < j$ are multi-defined and will be conventionally assigned the values 0 and $\top^{j-i}$, respectively.

The following tables summarize $\mid$ and $\setminus$. Here u, v, and z are arbitrary elements from Δ^+ satisfying $v = uz$ and such that $|u| = i$ and $|v| = j$.

$\mid$	0	ε	u	v	$\top^i$	$\top^j$
0	√					
ε	√	√	√	√	√	√
u	√		√	√		√
v	√			√		
$\top^i$	√				√	√
$\top^j$	√					√

$\setminus$	0	ε	u	v	$\top^i$	$\top^j$
0	(0)					
ε	0	ε	u	v	$\top^i$	$\top^j$
u	0		ε	z		$\top^j$
v	0			ε		
$\top^i$	0				ε	$(\top^{j-i})$
$\top^j$	0					ε

We can define the $\wedge$ of elements of Δ^* or from $\top^+$ as their greatest common prefix. The $\wedge$ of an element $u \in \Delta^*$ with an element $\top^i \in \{\top\}^+$ is

u truncated to $i-1$ characters if $|u| \geq i$. The $\wedge$ of 0 or $\top^i$ with $\top^i$ is $\top^i$ and $0 \wedge 0 = 0$.

The following table summarizes the $\wedge$ operation. Here u, v, and z are arbitrary elements from Δ^+ satisfying $v = uz$ and such that $|u| = i$ and $|v| = j$ and u' is the first $i-1$ characters of u and v' is the first $j-1$ characters of v.

$\wedge$	0	ε	u	v	$\top^i$	$\top^j$
0	0	ε	u	v	$\top^i$	$\top^j$
ε	ε	ε	ε	ε	ε	ε
u	u	ε	u	u	u'	u
v	v	ε	u	v	u'	v'
$\top^i$	$\top^i$	ε	u'	u'	$\top^i$	$\top^i$
$\top^j$	$\top^j$	ε	u	v'	$\top^i$	$\top^i$

Lemma 6.4 *$\wedge$ defines a GCLD operation with respect to $|$.*

The proof follows a case analysis similar to lemma 5.4.

This process can be summarized by the following algorithm for computing $\bigwedge A$ where A is a possibly infinite subset of $\mathbf{LM}(\Delta^*)$.

> If A has at least one element from Δ^* then $\bigwedge A$ is the longest common prefix of the Δ^* elements truncated to length $i-1$ where $\top^i$ is the shortest element in $\{\top\}^+ \cap A$. Otherwise if $\top^i \in A$ then $\bigwedge A$ is the shortest $\top^i$. Otherwise $A = \{0\}$ and $\bigwedge A = 0$.

Theorem 6.5 $\mathbf{LM}(\Delta^*)$ *is a GCLD $*$-semiring. Furthermore $\bigwedge$ is defined even for infinite sets.*

Theorem 3.1 and algorithm 4.2 can then be applied, to allow the computation of the shortest of the possible outputs for a given input. If a result from Δ^* is obtained then this is the output. If the value produced is a power of $\top$ then there remains an multiplicity of values and a domain error is reported. The overall effect is to disambiguate a transduction by selecting the shortest word and report an error is a multiplicity of outputs still remain.

7 Selecting the Genealogical Minimum

The third possibility involves a refinement of selecting the shortest output. If there is a tie, select that word which is earlier in lexicographic ordering. This will be achieved using the Genealogical-Minimum $*$-semiring $\mathbf{GM}(\Delta^*)$ together with theorem 3.1 and algorithm 4.2.

The *genealogical order* $\sqsubset$ on Δ^* is defined as follows: $u \sqsubset v$ if and only if either (1) $|u| < |v|$ or (2) $|u| = |v|$ and there are $x, u', v' \in \Delta^*$ and $a, b \in \Delta$

such that $u = xau'$, $v = xbv'$, and $a \prec b$ in some prespecified total order on Δ. Then $u \sqsubseteq v$ if and only if $u \sqsubset v$ or $u = v$.

The *genealogical minimum* χ of a set $S \subseteq \Delta^*$ is then defined as the unique value $w \in S$ which precedes all other values in S ($u \in S \Longrightarrow w \sqsubseteq u$). Since $\sqsubseteq$ is a well order, the minimum is defined for all non-empty subsets of Δ^* and always yields a single value. For convenience, we will define $\chi(\emptyset) = 0$, where 0 will indicate the absence of a value.

The algebra $\mathbf{GM}(\Delta^*) = \langle \Delta^* \cup \{0\}, \oplus, \odot, ^*, 0, \varepsilon \rangle$ will now be defined. The elements are those of Δ^* extended with the 0 element.

The multiplicative operation $\odot$ is that of the free monoid for elements from Δ^*. Multiplication of elements from Δ^* on either side by 0 yields 0. It is easily shown that $\odot$ is associative, has ε as a identity (left and right), and 0 as a zero (left and right).

The additive operation $\oplus$ applied to elements of Δ^* yields the genealogical minimum. The element 0 is a left and right identity. It is easily shown that $\oplus$ is associative, commutative, idempotent, and has 0 as an identity.

The star of any element is ε. The star property $((x^* \odot x) \oplus \varepsilon = (x \odot x^*) \oplus \varepsilon = x^*)$ is easily verified.

The following tables summarize $\oplus$, $\odot$, and $\star$. Here u and v are arbitrary distinct elements of Δ^+ and $z = \chi(\{u, v\})$:

$\oplus$	0	ε	u	v
0	0	ε	u	v
ε	ε	ε	ε	ε
u	u	ε	u	z
v	v	ε	z	v

$\odot$	0	ε	u	v
0	0	0	0	0
ε	0	ε	u	v
u	0	u	uu	uv
v	0	v	vu	vv

$*$	
0	ε
ε	ε
u	ε
v	ε

Lemma 7.1 *The distributive laws* $u \odot (v \oplus w) = (u \odot v) \oplus (u \odot w)$ *and* $(v \oplus w) \odot u = (v \odot u) \oplus (w \odot u)$ *hold for any* $u, v, w \in \mathbf{GM}(\Delta^*)$.

Proof: Since $v \oplus w$ must equal either v or w and addition is idempotent, both sides of the left distributive law reduce to $u \odot v$ or $u \odot w$, respectively. Similarly both sides of the right distributive law reduce to $v \odot u$ or $w \odot u$, respectively. □

Summing up these observations we have the following theorem.

Theorem 7.2 $\mathbf{GM}(\Delta^*)$ *is an idempotent* $*$*-semiring.*

Theorem 7.3 χ *is a* $*$*-semiring morphism.*

Proof: It is necessary to show that χ preserves the operations:

$\chi(\emptyset) = 0$: By definition.

$\chi(\{\varepsilon\}) = \varepsilon$: Immediate.

$\chi(u \cup v) = \chi(u) \oplus \chi(v)$: Both sides yield the genealogical minimum element from $u \cup v$.

$\chi(uv) = \chi(u) \odot \chi(v)$: The words of minimum length from uv are formed by concatenating the shortest words from u with the shortest words from v. Since all of the minimum length uv words then factor into u and v words at the same point, the genealogical minimum of the uv words will be exactly the genealogical minimum u words concatenated with the genealogical minimum v word.

$\chi(u^*) = \chi(u)^*$: Both sides equal ε.

□

If both u and v are from Δ^* then $u \mid v$ if and only if u is a (not necessarily proper) prefix of v. Then $u \setminus v$ equals the suffix of v which remains after u is removed. Any string divides 0.

The following tables summarize $\mid$ and $\setminus$. Here u, v, and z are arbitrary elements from Δ^+ satisfying $v = uz$.

$\mid$	0	ε	u	v
0	√			
ε	√	√	√	√
u	√		√	√
v	√			√

$\setminus$	0	ε	u	v
0	(0)			
ε	0	ε	u	v
u	0		ε	z
v	0			ε

We can define the $\wedge$ of elements of Δ^* as their greatest common prefix and the $\wedge$ of a $u \in \Delta^*$ with 0 is u.

The following table summarizes the $\wedge$ operation. Here u, v, and z are arbitrary elements from Δ^+ satisfying $v = uz$.

$\wedge$	0	ε	u	v
0	0	ε	u	v
ε	ε	ε	ε	ε
u	u	ε	u	u
v	v	ε	u	v

Lemma 7.4 *$\wedge$ defines a GCLD operation with respect to $\mid$.*

The proof follows a case analysis similar to lemma 5.4.

This process can be summarized by the following algorithm for computing $\bigwedge A$ where A is a possibly infinite subset from $\mathbf{GM}(\Delta^*)$.

If A has at least one element from Δ^* then $\bigwedge A$ is the longest common prefix of the Δ^* elements. Otherwise $A = \{0\}$ and $\bigwedge A = 0$.

Theorem 3.1 and algorithm 4.1 can then be applied, to allow the computation of the genealogical minimum of the possible outputs for a given input. That is for input w and finite transducer T, we can compute $\chi(|T|(w))$.

8 Selecting the Lexicographic Minimum

The fourth approach involves selecting the lexicographic minimum element. This will be achieved using the Lexicographic-Minimum $*$-semiring $\mathbf{XM}(\Delta^*)$.

The *lexicographic order* $\prec_l$ on Δ^* is defined as follows: $u \prec_l v$ if and only if either (1) u is a prefix of v or (2) there are $x, u', v' \in \Delta^*$ and $a, b \in \Delta$ such that $u = xau'$, $v = xbv'$, and $a \prec b$ in some prespecified total order on Δ. Then $u \preceq_l v$ if and only if $u \prec_l v$ or $u = v$.

It will be convenient for our purposes to define a partial order $\prec$ as the second part of this standard definition: $u \prec v$ if and only if it is possible to write $u = xau'$ and $v = xbv'$ where $x, u', v' \in \Delta^*$, and $a, b \in \Delta$ where $a \prec b$. $u \preceq v$ if $u \prec v$ or $u = v$.

Then we will define a mapping ψ which maps a set of words S over Δ^* to the largest subset which are not preceded by other words in S:

$u \in \psi(S)$ if and only if $u \in S$ and there is no $v \in S$ such that $v \prec u$.

Note that then the following will always hold:

If u, v belong to $\psi(S)$ then either u is a prefix of v or v is a prefix of u.

Thus all of the words in $\psi(S)$ will be prefixes of some possibly infinite word.

Lemma 8.1 *If $R \in \mathbf{Reg}(\Delta^*)$ then $\psi(R)$ is an effectively computable regular set.*

Proof: If R is effectively presented, then we can construct a deterministic finite automaton for it. If we then discard all transitions from any state which correspond to letters which are not the lexicographic minimum letters exiting that state. Then we can discard any states which are not accessible from the start state of from which a final state cannot be reached. The resulting trim DFA then recognizes $\psi(R)$. To see this, note that it satisfies the above prefix property since every state has only one successor so that all words in the set must be a prefix of some, possibly infinite, word. Secondly, words are deleted as a result of removing transitions, and it is easy

to see that every word removed as a result of breaking a particular transition is lexicographically greater than any word obtained by taking the one remaining outgoing transition instead of the broken one. □

Note that the structure of the resulting deterministic finite automaton will have one of two forms:

1. If the longest word in $\psi(S)$ is finite, then the DFA will be a string of states which recognizes this word with other states marked as final to indicate the selected prefixes.

2. If there is no longest word in $\psi(S)$, then the DFA will be a string of states that loops back on itself with at least one of the states in the loop marked as final.

Thus although the possible structures for $\psi(S)$ is not as simple as in the $*$-semirings introduced in preceding sections, it is still the case that it is simpler than $\mathbf{Reg}(\Delta^*)$.

Before we introduce the $*$-semiring $\mathbf{XM}(\Delta^*)$ it will be useful to show a number of technical lemmas. These together will show that we can drop any ψ's except the last from an expression involving $\cup$, $\cdot$, *, and ψ. These results hold for any language although we will subsequently restrict attention to regular languages.

Lemma 8.2 *$\psi(a \cup b) = \psi(\psi(a) \cup b) = \psi(a \cup \psi(b))$ for all $a, b \subseteq \Delta^*$.*

Proof: We will prove directly $\psi(a \cup b) = \psi(\psi(a) \cup b)$. The equality $\psi(a \cup b) = \psi(a \cup \psi(b))$ will follow from commutativity of $\cup$.

($\subseteq$) Let $x \in \psi(a \cup b)$. Then $x \in a \cup b$ and there is no $y \in a \cup b$ such that $y \prec x$. If $x \in a$ then $x \in \psi(a)$ since there is no $y \in a$ such that $y \prec x$. Thus $x \in \psi(a) \cup b$. Since there is no $y \in \psi(a) \cup b$ such that $y \prec x$, $x \in \psi(\psi(a) \cup b)$.

($\supseteq$) Let $x \in \psi(\psi(a) \cup b)$. Then $x \in \psi(a) \cup b$ and there is no $y \in \psi(a) \cup b$ such that $y \prec x$. Then $x \in a \cup b$ and the only way x could fail to be in $\psi(a \cup b)$ is if there were a z in $a - \psi(a)$ such that $z \prec x$. But then there would be some $y \in \psi(a)$ such that $y \prec z$ and thus by transitivity, this y in $\psi(a) \cup b$ satisfies $y \prec x$. This contradicts the hypothesis. □

Lemma 8.3 *$\psi(ab) = \psi(\psi(a)b)$ for all $a, b \subseteq \Delta^*$.*

Proof: ($\subseteq$) Let $x \in \psi(ab)$. Then $x \in ab$ and there is no $y \in ab$ such that $y \prec x$. Then x must be in $\psi(a)b$ since otherwise $x = x_1x_2$ where $x_1 \in a - \psi(a)$ and $x_2 \in b$, so that there would be a $z \in \psi(a)$ satisfying $z \prec x_1$, and $zx_2 \prec x$. Since $x \in \psi(a)b$ and there is no $y \in \psi(a)b$ such that $y \prec x$ it must be the case that $x \in \psi(\psi(a)b)$.

($\supseteq$) Let $x \in \psi(\psi(a)b)$. Then $x \in \psi(a)b$ and there is no $y \in \psi(a)b$ such that $y \prec x$. Then $x \in ab$ and $x \in \psi(ab)$ unless there is a $y \in (a - \psi(a))b$

satisfying $y \prec x$. But then $y = y_1 y_2$ where $y_1 \in a - \psi(a)$ and $y_2 \in b$ and there must be a $z \in \psi(a)$ such that $z \prec y_1$. Since then $zy_2 \prec x$ our hypothesis is contradicted. □

Lemma 8.4 *$\psi(ab) = \psi(a\psi(b))$ for all $a, b \subseteq \Delta^*$.*

Proof: ($\subseteq$) Let $x \in \psi(ab)$. Then $x \in ab$ and there is no $y \in ab$ such that $y \prec x$. Then x must be in $a\psi(b)$ since otherwise $x = x_1 x_2$ where $x_1 \in a$ and $x_2 \in b - \psi(b)$, so that there would be a $z \in \psi(b)$ satisfying $z \prec x_2$, and $x_1 z \prec x$. Since $x \in a\psi(b)$ and there is no $y \in a\psi(b)$ such that $y \prec x$ it must be the case that $x \in \psi(a\psi(b))$.

($\supseteq$) Let $x \in \psi(a\psi(b))$. Then $x \in a\psi(b)$ and there is no $y \in a\psi(b)$ such that $y \prec x$. Then $x \in ab$ and $x \in \psi(ab)$ unless there is a $y \in a(b - \psi(b))$ satisfying $y \prec x$. But then $y = y_1 y_2$ where $y_1 \in a$ and $y_2 \in b - \psi(b)$ and there must be a $z \in \psi(b)$ such that $z \prec y_2$. Since then $y_1 z \prec x$ our hypothesis is contradicted. □

Lemma 8.5 *$\psi(a^*) = \psi(\psi(a)^*)$ for all $a \subseteq \Delta^*$.*

Proof: ($\subseteq$) Let $x \in \psi(a^*)$. Then $x \in a^*$ and there is no $y \in a^*$ such that $y \prec x$. Then x must be in $\psi(a)^*$ since otherwise $x = x_1 x_2 \cdots x_k$ where $x_1, x_2, \ldots, x_{i-1} \in \psi(a)$, $x_i \in a - \psi(a)$, and $x_{i+1}, \ldots, x_k \in a$ for some $i \geq 1$, so that there would be a $z \in \psi(a)$ satisfying $z \prec x_i$, and $x_1 x_2 \cdots x_{i-1} z \prec x$. Since $x \in \psi(a)^*$ and there is no $y \in \psi(a)^*$ such that $y \prec x$ it must be the case that $x \in \psi(\psi(a)^*)$.

($\supseteq$) Let $x \in \psi(\psi(a)^*)$. Then $x \in \psi(a)^*$ and there is no $y \in \psi(a)^*$ such that $y \prec x$. Then $x \in a^*$ and $x \in \psi(a^*)$ unless there is a $y \in \psi(a)^{i-1}(a - \psi(a))a^*$ satisfying $y \prec x$. But then $y = y_1 y_2 \cdots y_k$ where $y_1, y_2, \ldots, y_{i-1} \in \psi(a)$, $y_i \in a - \psi(a)$, and $y_{i+1}, \ldots, y_k \in a$ for some $i \geq 1$ and there must be a $z \in \psi(a)$ such that $z \prec y_i$. Since then $y_1 y_2 \cdots z \prec x$ our hypothesis is contradicted. □

The algebra $\mathbf{XM}(\Delta^*) = \langle \psi(\mathbf{Reg}(\Delta^*)), \oplus, \odot, ^*, \emptyset, \{\varepsilon\} \rangle$ where $\psi(\mathbf{Reg}(\Delta^*))$ is the set of regular sets over Δ^* in the image of φ, and $\oplus$, $\odot$, and * are the images of $\cup$, $\cdot$, and * respectively.

$$a \oplus b = \psi(a \cup b) \qquad a \odot b = \psi(ab) \qquad a^* = \psi(a^*) \qquad \forall a, b \in \psi(\mathbf{Reg}(\Delta^*))$$

Theorem 8.6 $\mathbf{XM}(\Delta^*)$ *is an idempotent $*$-semiring.*

Proof: This follows from the lemmas 8.2, 8.3, 8.4, and 8.5 and the observation that that $\mathbf{Reg}(\Delta^*)$ is an idempotent $*$-semiring. For example, to verify the left distributive law we can write:

$$u \odot (v \oplus w) = \psi(u\psi(v \cup w)) = \psi(u(v \cup w))$$

$$= \psi(uv \cup uw) = \psi(\psi(uv) \cup \psi(uw)) = (u \odot v) \oplus (u \odot w)$$

Similar arguments work for each of the other 9 laws. □

Theorem 8.7 *ψ is a $*$-semiring morphism.*

Proof: It is necessary to show that this mapping preserves the operations of a $*$-semiring:

$\psi(\emptyset) = \emptyset$: Immediate.

$\psi(\{\varepsilon\}) = \{\varepsilon\}$: Immediate.

$\psi(u \cup v) = \psi(u) \oplus \psi(v)$: Immediate from $\psi(u \cup v) = \psi(\psi(u) \cup \psi(v))$.

$\psi(uv) = \psi(u) \odot \psi(v)$: Immediate from $\psi(uv) = \psi(\psi(u)\psi(v))$.

$\psi(u^*) = \psi(u)^*$: Immediate from $\psi(u^*) = \psi(\psi(u)^*)$.

□

Theorem 8.8 $\mathbf{XM}(\Delta^*)$ *is not a GCLD $*$-semiring.*

Proof: Let $\Delta = \{a, b, c\}$ with chosen order $a \prec b \prec c$ and consider the sets $u = \{\varepsilon, a\}$ and $v = \{\varepsilon, b\}$. Then u and v are both divisible by the $\{\varepsilon, c\}$ and $\{\varepsilon, cc\}$, neither of which divides the other.

$$\{\varepsilon, c\} \odot \{\varepsilon, a\} = \{\varepsilon, a\} \qquad \{\varepsilon, c\} \odot \{\varepsilon, b\} = \{\varepsilon, b\}$$

$$\{\varepsilon, cc\} \odot \{\varepsilon, a\} = \{\varepsilon, a\} \qquad \{\varepsilon, cc\} \odot \{\varepsilon, b\} = \{\varepsilon, b\}$$

If $w = \{\varepsilon, c\} \odot x$, however, x must contain ε so that w must contain c or a word beginning with a or b. Thus $\{\varepsilon, c\} \not| \{\varepsilon, cc\}$. Similarly, we can show $\{\varepsilon, cc\} \not| \{\varepsilon, c\}$. □

It will be useful for our purposes to let $u \tilde{\wedge} v$ to be the greatest common prefix of the shortest words from u and v. Clearly the operation $\tilde{\wedge}$ is uniquely defined and satisfies the commutative and associative laws.

Using theorem 3.1 and algorithm 4.2 with $*$-semiring $\mathbf{XM}(\Delta^*)$ will yield, for a given input w and finite transducer T, a set of words as output. To extract the true lexicographic minimum all that is needed is the selection of the shortest word in this set. This can be achieved technically, by introducing a new symbol $\dashv$ which designates end of input and precedes all other letters in the $\prec$ ordering.

9 Implementing Elseor

One of the original motivations for this research is a quest for a good way of implementing the following operation for finite transductions. Let f and g be finite transductions. Define $h = f \,||\, g$ to be the mapping which satisfies

$$h(w) = \begin{cases} f(w) & \text{if } w \in \text{dom}(f) \\ g(w) & \text{if } w \in \text{dom}(g) - \text{dom}(f) \end{cases}$$

This operation has been called "elseor" because of it similarity to a union operation and because it is used in situations where programming languages would use "elseif". For example, the following fragment also describes $f \mid\mid g$:

if $w \in \mathrm{dom}(f)$ **then return** $f(w)$
elseif $w \in \mathrm{dom}(g)$ **then return** $g(w)$
else fail

The following is another definition which shows that $f \mid\mid g$ is a finite transduction if f and g are:

$$f \mid\mid g = f \cup (((\mathrm{dom}(g) - \mathrm{dom}(f)) \times \mathrm{ran}(g)) \cap g)$$

Here "dom" and "ran" indicate the domain and range, respectively. This shows that $f \mid\mid g$ is a finite transduction if f and g are since (1) $\mathrm{dom}(f)$ and $\mathrm{dom}(g)$ are regular languages, (2) $\mathrm{dom}(g) - \mathrm{dom}(f)$ is then regular, (3) the domain restriction of a finite transduction is a finite transduction, and (4) the union of two finite transductions is a finite transduction. This computation is in fact performed by INR, a program implemented by the author for constructing large finite transducers [Joh86]. However, this is not a very good solution, since experience has shown that this approach leads to state explosion in transducers which are otherwise manageable.

Since elseor is extremely useful for describing single-valued transductions in terms of more primitive ones, it is worth the effort of looking for a better solution. So the question is how to mark the transducer minimally so that the process can still be performed. The last section gives us most of the answer. Suppose we wish to compute $f \mid\mid g$ as some subcomponent of a finite transducer. Then we can replace this expression by $(\varepsilon, \mathcal{A})f \cup (\varepsilon, \mathcal{B})g$, that is, we introduce new letters $\mathcal{A}$ and $\mathcal{B}$, precede f by $\mathcal{A}$, and g by $\mathcal{B}$, replace $\mid\mid$ by ordinary union. The new transducer will then write $\mathcal{A}$ if the f alternative is possible and $\mathcal{B}$ if the g alternative is possible. Then if we select the lexicographic minimum output assuming $\mathcal{A} \prec \mathcal{B}$, and suppress the $\mathcal{A}$ and $\mathcal{B}$ we will have the answer we desire.

The main problem with this is that choosing the lexicographic minimum is going to resolve ambiguities which we would rather report as errors. We also have to consider more precisely the role of the $\mathcal{A}$, $\mathcal{B}$ letters.

Let $\prec$ then be a partial order on the alphabet Δ and define the partial order $\prec$ on Δ^* as before: $u \prec v$ if and only if it is possible to write $u = xau'$ and $v = xbv'$ where $x, u', v' \in \Delta^*$, and $a, b \in \Delta$ where $a \prec b$. $u \preceq v$ if $u \prec v$ or $u = v$.

Then, as before, we will define a mapping ψ which maps a set of words S over Δ^* to the largest subset which are not preceded by other words in S:

$u \in \psi(S)$ if and only if $u \in S$ and there is no $v \in S$ such that $v \prec u$.

Based on this we will define a mapping θ which is equal to ψ if it maps to an element satisfying the prefix property:

If u, v belong to $\psi(S)$ then either u is a prefix of v or v is a prefix of u.

Otherwise θ will map to the element $\top$. Thus if not equal to $\top$ all of the words in $\theta(S)$ will be prefixes of some possibly infinite word.

Lemma 9.1 *If $R \in \mathbf{Reg}(\Delta^*)$ then $\theta(R)$ is an effectively computable regular set.*

Proof: As before, $\psi(R)$ is effectively computable. From a trim deterministic finite automaton, it is possible to test whether it satisfies the prefix property. □

It can then be shown that θ is a $*$-semiring morphism. A technique based on theorem 3.1, algorithm 4.2, and the comments at the end of section 8 will achieve the desired goal.

10 Minimum Delay Recognition

We will say that an algorithm is *minimum delay* if after reading any prefix of the input, it has written the $\bigwedge$ of the outputs which can be produced for inputs possibly beginning with that prefix. This is a property of the behaviour of the automaton rather than the automaton itself.

If we denote by $\pi(u)$ the minimum delay output for a prefix u then

$$\pi(u) = \bigwedge\{|T|(uv) \mid v \in \Sigma^*\} \qquad \pi(u\dashv) = |T|(u).$$

This leads to the following *on-line* algorithm:

Algorithm 10.1

$x : \Sigma$; $u : \Sigma^*$;
write$(\pi(\varepsilon))$;
while true do begin
  $x :=$ *read_character*;
  write$(\pi(u) \setminus \pi(ux))$;
  $u := ux$;
  if $x = \dashv$ **then exit loop**
end;

Let σ be a diagonal matrix satisfying

$$\sigma_{ii} = \bigwedge\{|T_i|(v) \mid v \in \Sigma^*\}$$

where $T_i = \langle \Sigma, K, \delta, s_i, F \rangle$ is formed from T by replacing s by a row vector with **1** in the i position and **0** in other locations. The matrix σ^{-1} is the diagonal matrix formed by taking the formal inverses of the corresponding elements from σ. Note the this matrix is only used in the context $\sigma^{-1}\hat{\delta}(x)\sigma$ and so no inverse elements will be introduced by this process.

Algorithm 10.2

$x : \Sigma; \quad z : K; \quad v : K^{1\times m}$
$z := \bigwedge \hat{s}\sigma;$
write(z);
$v := z \setminus \hat{s}\sigma;$
$x :=$ *read_character*;
while $x \neq \dashv$ **do begin**
    $z := \bigwedge v\sigma^{-1}\hat{\delta}(x)\sigma;$
    write(z);
    $v := z \setminus v\sigma^{-1}\hat{\delta}(x)\sigma;$
    $x :=$ *read_character*
end;
write$(v\sigma^{-1}F)$;

Suppose that we are given a $\mathbf{Reg}(\Delta^*)$-NFA. If we are only interested in the answer to a transduction when it is single-valued, we can convert it into a $\mathbf{SV}(\Delta^*)$-NFA which yields a single value exactly when the original does. This follows from theorem 3.1.

However, σ is, in general, not effectively computable as the following theorem shows:

Theorem 10.1 *Let T be a $\mathbf{SV}(\Delta^*)$-NFA. Then σ is not effectively computable.*

Proof: Let $u_1, u_2, \ldots, u_p,\ v_1, v_2, \ldots, v_p$ be an instance of Post's Correspondence Problem. Then the following rational relations can be constructed:

$$U = \{(ab, u_1), (a^2b, u_2), \ldots, (a^pb, u_p)\}$$

$$V = \{(ab, v_1), (a^2b, v_2), \ldots, (a^pb, v_p)\}$$

$$R = ((\varepsilon, a)(U^+ \cup V^+)) \cup (\varepsilon, b)$$

If the instance of PCP has a solution then there is a non-ε input w which yields a single result. This result will begin with a and so $\pi(\varepsilon) = \varepsilon$. If the

instance of PCP has no solution then every input will yield a multiple result and $\pi(\epsilon) = b$. If an automaton is constructed with a state i which precedes this set then σ_{ii} will not be computable. □

On the other hand, if we restrict attention to rational functions, that is, to $\mathbf{SV}(\Delta^*)$-NFA's which never produce $\top$, the process becomes effective.

Theorem 10.2 *Let T be a $\mathbf{SV}(\Delta^*)$-NFA. Then it is decidable whether there is an input w such that $|T|(w) = \top$. Furthermore, if T cannot produce $\top$, then σ can be effectively computed.*

Proof: The decidability of whether T can emit $\top$ is reducible to the decidability of whether a rational transduction is single-valued. This is known to be decidable [BH77,Ber79].

If $\top$ cannot be output the value of σ_{ii} can be computed by computing $\wedge$ of the regular set of outputs possible from i. □

11 Deterministic Left to Right Recognition

After any prefix of the input is read by algorithm 10.2, the vector v will contain a vector of K values. This information is all that is required to completely characterize the "state" or current configuration of the computation. It is possible to define an infinite graph of all configurations connected by deterministic transitions. The accessible part of this infinite graph is that part which can be reached from the beginning configuration, that is, the configuration $\bigwedge \hat{s}\sigma$. If finite this process defines a K-DFA equivalent to the original K-NFA.

This is summarized in the following algorithm. Here $index(v)$ is a function which maps each configuration v to a distinct index value. Conceptually, it works by determining whether a configuration has ever been seen before and returning the same index if it has. Otherwise an unused index is found and returned. Every index is assumed to be unconsidered as soon as it is used until it is explicitly considered. The function $config(i)$ returns the configuration associated with a particular index. This algorithm takes a K-NFA $\langle \Sigma, K, \delta, s, F \rangle$ and, if it terminates, produces an equivalent K-DFA $\langle \Sigma, K, \delta', s', F' \rangle$.

Algorithm 11.1

$x : \Sigma; \quad z : K; \quad v : K^{1\times m}$
$s' := 0;$
$z := \bigwedge \hat{s}\sigma;$
$v := z \setminus \hat{s}\sigma;$
$i := index(v);$

$s'[i] := z;$
while *not all indexes considered* **do begin**
    $i :=$ *any unconsidered index;*
    $v :=$ *config*(i)*;*
    for $x \in \Sigma$ **do begin**
        $z := \bigwedge v\sigma^{-1}\hat{\delta}(x)\sigma;$
        $j :=$ *index*$(z \setminus v\sigma^{-1}\hat{\delta}(x)\sigma);$
        $\delta'(x)_{ij} := z;$
        $F'(i) := v\sigma^{-1}F$
end end;

This algorithm will run forever if there are an infinite number of accessible configurations. A much better state of affairs would be to detect this condition after a finite amount of time, report the condition, and quit. One such test is based on Choffrut's twinning property [Cho77,Ber79].

The resulting machine may have more states than necessary for the same reason that the accessible subsets construction yields a non-optimal deterministic finite automaton. It is possible, however, to use the same solution based on Nerode's theorem.

The technique involves constructing a conventional Deterministic Finite Automaton over an alphabet formed from an appropriate finite subset of $\Sigma \times K$. This machine will have a transition from i to j with label (x, k) if and only if $\delta(x)_{ij} = k$. Clearly this operation is well-defined and invertible and only a finite alphabet will be used. Furthermore if the process is inverted on any other automaton recognizing the same regular set, then an equivalent K-DFA will be produced, i.e., one with the same behaviour. Thus one can apply the usual DFA minimization, and invert to yield an automaton with no more and probably fewer states.

The algorithm thus described, that is, algorithm 11.1 followed by the minimization step will be called algorithm D and will be applicable to any GCLD $*$-semiring.

12 Subsequential Transduction

The construction of section 11 may be used to compute a subsequential transducer from a description for a rational function if possible. Furthermore in a real sense this transducer will be optimal. Suppose that we are given a $\mathbf{SV}(\Delta^*)$-NFA T and if algorithm D terminates successfully a subsequential transducer S is produced.

Theorem 12.1 *Algorithm D may be used to compute a minimum delay minimum state subsequential transducer for any subsequential function pre-*

sented as a rational function.

Example 12.1 *Let* $T = \langle \{x\}, \mathbf{SV}(\{t,z,y\}^*), \delta, s, F \rangle$ *satisfy:*

$$s = \begin{bmatrix} \varepsilon & 0 & 0 & 0 \end{bmatrix} \qquad \delta(x) = \begin{bmatrix} 0 & tzy & tz & tzy \\ 0 & 0 & z & 0 \\ 0 & z & 0 & y \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad F = \begin{bmatrix} \varepsilon \\ 0 \\ 0 \\ \varepsilon \end{bmatrix}$$

Then

$$\sigma = \begin{bmatrix} \varepsilon & 0 & 0 & 0 \\ 0 & zy & 0 & 0 \\ 0 & 0 & y & 0 \\ 0 & 0 & 0 & \varepsilon \end{bmatrix}$$

The accessible configurations then are:

$$\begin{aligned} v_1 &= \begin{bmatrix} \varepsilon & 0 & 0 & 0 \end{bmatrix} \\ v_2 &= \begin{bmatrix} 0 & zy & \varepsilon & \varepsilon \end{bmatrix} \\ v_3 &= \begin{bmatrix} 0 & zy & zy & \varepsilon \end{bmatrix} \end{aligned}$$

yielding the result $T' \langle \{x\}, \mathbf{SV}(\{t,z,y\}^*), \delta', s', F' \rangle$

$$s' = \begin{bmatrix} \varepsilon & 0 & 0 \end{bmatrix} \qquad \delta'(x) = \begin{bmatrix} 0 & tzy & 0 \\ 0 & 0 & \varepsilon \\ 0 & zy & 0 \end{bmatrix} \qquad F' = \begin{bmatrix} \varepsilon \\ \varepsilon \\ \varepsilon \end{bmatrix}$$

There are no equivalent states so that the minimization will not decrease the number of states.

The main result of this section will be proven in several steps. Here T will indicate the input to algorithm D and S will indicate the output.

Lemma 12.2 *If algorithm D terminates then* $|S| \subseteq |T|$.

Proof: Suppose $(u,v) \in |S|$. Then there is a path through S with label (u,v). Each of the states visited corresponds to a configuration vector of values from $\mathbf{SV}(\Delta^*)$. A path through T with the same label can be found by tracing backwards from the final state of S using a "how did I get here" kind of analysis. □

Lemma 12.3 *If algorithm D terminates then* $|T| \subseteq |S|$.

Proof: Suppose $(u,v) \in |T|$. Then there is a path through T with label (u,v). There must then be a sequence of transitions with input label u using a forward induction. □

Lemma 12.4 *If T is trim and $|T|$ is subsequential then algorithm D terminates.*

Proof: If T is subsequential then there is a subsequential transducer S' such that $|S'| = |T|$. Consider the states reached in S' and the possibly infinite machine S produced by algorithm 11.1 after reading some word w. Suppose S' is in state q and S is in configuration v. Consider the longest word in v. Suppose it is associated with state i from T. Since T is trim, there must be a path from i to a final state, say with input label z. This path can be chosen so that no state in T is repeated so that $|z| \leq m$. Thus from q the word z must also lead to a final state with the same output. Suppose that all strings occurring in the δ matrices and F vector for S' are bounded by a constant k. Then the additional output generated by S' is bounded by km. Since S can never be behind S' in the quantity of output generated, the ith component of v must also be less than km. Since the ith component was the longest, all components must be shorter than km. But there are only a finite number of configurations with no string exceeding a given bound. Thus S can have only a finite number of configurations and so algorithm 11.1 must terminate. Thus algorithm D must terminate. □

Lemma 12.5 *If T_1 and T_2 are two* $\mathbf{SV}(\Delta^*)$*-NFA'a satisfying $|T_1| = |T_2|$, then algorithm D yields subsequential transducers S_1 and S_2 that are identical up to renaming of states.*

Proof: Note that the transitions produced from any state depend only on the function and not on the particular transducer provided. Thus the only variation that can occur in algorithm 11.1 is that two or more states may be created which have the same outgoing transitions and can occur in the same right context. These will be combined in the minimization step. □

Lemma 12.6 *Algorithm D yields a subsequential transducer which has the smallest number of states. Furthermore this transducer has the minimum delay property and is canonical.*

Proof: Suppose that algorithm D is given a transducer T and a minimum state subsequential transducer S for T. Since S is deterministic every accessible configuration will contain exactly one state and the coefficient must be ε. Thus a subsequential transducer with no more states than S will be produced. Since algorithm D always yields an isomorphic answer for inputs having the same behaviour, it must yield a minimum state subsequential transducer for T as well. □

13 Conclusions and Further Work

The techniques outlined in the previous sections seem to be quite general and unify a collection of ideas. The mechanisms of $*$-semirings and K-automata allow several techniques for disambiguating (i.e., making single-valued) finite transductions and provide new insights into rational and subsequential functions.

However, there are still a number of questions which remain:

1. The $*$-semiring $\mathbf{Reg}(\Delta^*)$ is in some senses the simplest to study but there remain a number of questions. For example, is it a GCLD $*$-semiring?

2. The techniques were described as pure strategies. In practice, however, a number of techniques need to be combined. How should this best be achieved?

3. Are there any other $*$-semirings which are useful?

4. What are the best ways of actually implementing these algorithms?

14 Acknowledgements

I wish to thank G. H. Gonnet, F. W. Tompa, and the New Oxford English Dictionary project for providing the motivation for this research and for many fruitful discussions. I also wish to thank D. Wood for his helpful comments.

References

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass, 1974.

[AS85] S. Kamal Abdali and B. David Saunders. Transitive closure and related semiring properties via eliminants. *Theoretical Computer Science*, 40:257–274, 1985.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, Reading, Mass, 1986.

[AU72] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation and Compliling, Vol. I: Parsing.* Prentice-Hall, Englewood Cliffs, N.J., 1972.

[BC75] R. C. Backhouse and B. A. Carré. Regular algebra applied to path-finding problems. *J. Inst. Math. Appl.*, 15:161–186, 1975.

[Ber79] Jean Berstel. *Transductions and Context-Free Languages.* B. G. Teubner, Stuttgart, Germany, 1979.

[BH77] Meera Blattner and Tom Head. Single valued a-transducers. *Journal of Computer and System Sciences*, 15:310–327, 1977.

[Car79] B. A. Carré. *Graphs and Networks.* Clarendon Press, Oxford, 1979.

[CC73] Karel Culik II and Rina Cohen. LR-Regular grammars—an extension of LR(k) grammars. *Journal of Computer and System Sciences*, 7(1):66–96, 1973.

[Cho77] Christian Choffrut. Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles. *Theoretical Computer Science*, 5:325–338, 1977.

[EM65] C. C. Elgot and J. E. Mezei. On relations defined by generalized finite automata. *IBM Journal of Research*, 9:47–65, 1965.

[Gin66] Seymour Ginsburg. *The Mathematical Theory of Context-Free Languages.* McGraw-Hill, New York, 1966.

[Har78] M. A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, Reading, Masss., 1978.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Mass, 1979.

[Joh83] J. Howard Johnson. *Formal Models for String Similarity.* PhD thesis, University of Waterloo, 1983. Available as University of Waterloo Research Report CS–83–32 (128 pages).

[Joh86] J. Howard Johnson. INR: a program for computing finite automata. 1986. Manuscript (24 pages).

[KS86] Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages.* Springer-Verlag, Berlin, 1986.

[Leh77] D. J. Lehmann. Algebraic structures for transitive closure. *Theoretical Computer Science*, 4(1):59–76, 1977.

[LS78] Mike E. Lesk and E. Schmidt. *Lex—a Lexical Analyzer Generator.* Murray Hill, NJ, 1978.

[Sch75] M. P. Schützenberger. Sur les relations rationelles. In *Automata theory and formal languages: 2nd GI Conference*, pages 209–213, 1975.

[Sch77] M. P. Schützenberger. Sur une variante des fonctions séquentielles. *Theoretical Computer Science*, 4:47–57, 1977.

[SS78] Arto Salomaa and Matti Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer-Verlag, New York, 1978.