Making Copies

Jeffrey Heinz

(joint work with Hossep Dolatian)

MIT
April 23, 2021

This Talk

1. Making copies is less complex than recognizing copies.
2. Consequently, we have a better understanding of how reduplicative processes in natural language fit with morpho-phonological processes generally.
3. Mathematical linguistics has a bright future in the 21st century.

`https://doi.org/10.15398/jlm.v8i1.245`
Dolatian and Heinz (2020)

Part I

Classifying Transformations

Doing Linguistic Typology

Requires two books:

- "encyclopedia of categories"
- "encyclopedia of types"

Wilhelm Von Humboldt

Encyclopedia of Types

Morphological Transformations

1. Null affixation
2. Prefixation
3. Suffixation
4. Circumfixation
5. Infixation
6. Truncation
7. Root and pattern
8. Umlaut/Ablaut
9. Partial Reduplication
10. Total Reduplication
11. ...

Item and Process

As a first approximation, we may think of morphological and phonological processes as string-to-string functions.

$$\text{kæt} \xrightarrow{\texttt{Pl}} \text{kæt-z} \xrightarrow{\texttt{Phon}} \text{kæts}$$

(English)

Item and Process

As a first approximation, we may think of morphological and phonological processes as string-to-string functions.

$$\text{hiiga} \xrightarrow{\texttt{1sg Hab}} \text{m-hiiga} \xrightarrow{\texttt{Phon}} \text{mpiiga}$$

('hunt', Kerewe)

Item and Process

As a first approximation, we may think of morphological and phonological processes as string-to-string functions.

$$\text{wanita} \xrightarrow{\texttt{Pl}} \text{wanita-wanita} \xrightarrow{\texttt{Phon}} \text{wanitawanita}$$

(‘woman’, Indonesian)

Item and Process

As a first approximation, we may think of morphological and phonological processes as string-to-string functions.

$$\text{takki} \xrightarrow{\texttt{Pl}} \text{tak-takki} \xrightarrow{\texttt{Phon}} \text{taktakki}$$

('leg', Agta)

Encyclopedia of Categories?

$$f : \Sigma^* \to \Delta^*$$

Questions

1. What is a 'local' transformation?
2. What are 'non-local' transformations?
3. What kinds of transformations require a lot of memory and/or computational resources?
4. What kinds of transformations do not?

Analogy to Real Functions

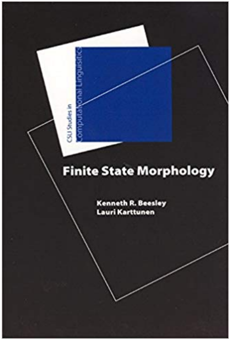
$$f : \mathbb{R} \to \mathbb{R}$$

Encyclopedia of Categories

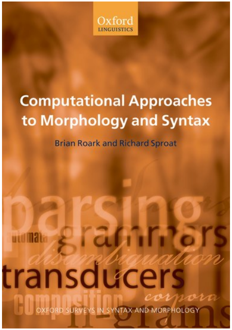
1. Linear functions
2. Step functions
3. Polynomial functions (quadratic, cubic, degree n)
4. Exponential functions
5. Logarithmic functions
6. Trigonometric functions (sin, tanh, ...)
7. ...

The established, foundational view

Word formation processes are rational relations, analyzable with (1-way) finite-state methods.



Beesley and Karttunen 2003



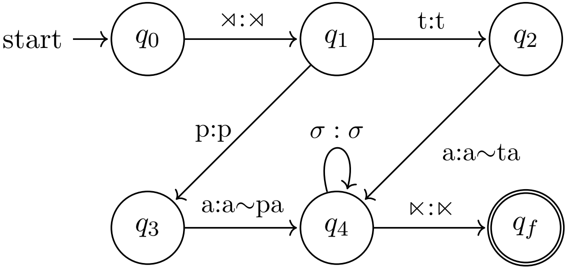
Roark and Sproat 2007

String-to-String Functions

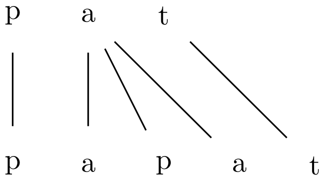
The established, foundational view

Rational Relations	non-Rational Relations
Prefixation	Total Reduplication
Suffixation	
Circumfixation	
Infixation	
Truncation	
Root and pattern	
Umlaut/Ablaut	
Partial Reduplication	
...	

Example 1way Finite-State Transducer for Partial Reduplication



$\text{pat} \xrightarrow{\texttt{CV-RED}} \text{pa-pat}$



String-to-String Functions

The established, foundational view

Rational Relations	non-Rational Relations
Prefixation	Total Reduplication
Suffixation	
Circumfixation	
Infixation	
Truncation	
Root and pattern	
Umlaut/Ablaut	
Partial Reduplication	
...	

String-to-String Functions

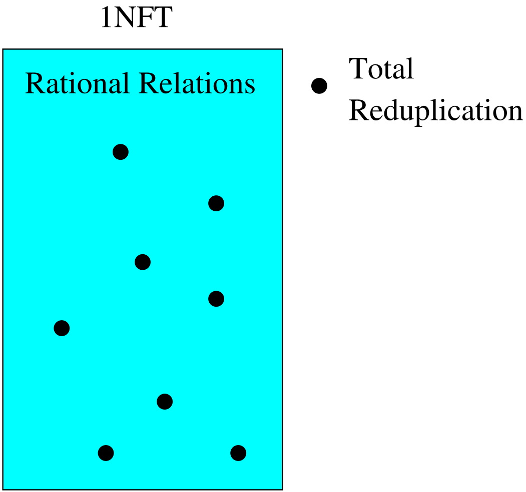
The established, foundational view

Rational Relations	non-Rational Relations
Prefixation	Total Reduplication
Suffixation	
Circumfixation	
Infixation	
Truncation	
Root and pattern	
Umlaut/Ablaut	
Partial Reduplication	
...	

I think linguistics will be well-served by a more articulated view of this kind of encyclopedia of categories. Formal language theory is not static! Much more to discover.

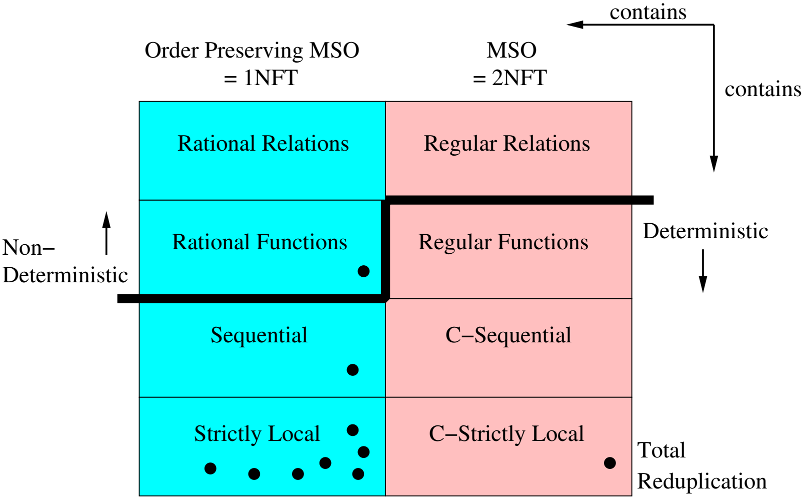
String Relations

The established, foundational view pictorially



String Relations

A more articulated view



(Filiot and Reynier 2016, Chandlee 2017, Dolatian and Heinz 2020)

Rational vs. Regular

For stringsets (formal languages) there is no distinction.

$$\llbracket \text{1DFA} \rrbracket = \llbracket \text{1NFA} \rrbracket = \llbracket \text{2NFA} \rrbracket$$

$$= \llbracket \text{RE} \rrbracket = \llbracket \text{GRE} \rrbracket$$

$$= \llbracket \text{MSO}(+1) \rrbracket = \llbracket \text{MSO}(<) \rrbracket$$

1/2	1-way or 2-way
N/D	Non-deterministic or Deterministic
FA	Finite-state Acceptor
(G)RE	(Generalized) Regular Expressions
MSO	Monadic Second Order with successor (+1) or precedence (<)

Rational vs. Regular

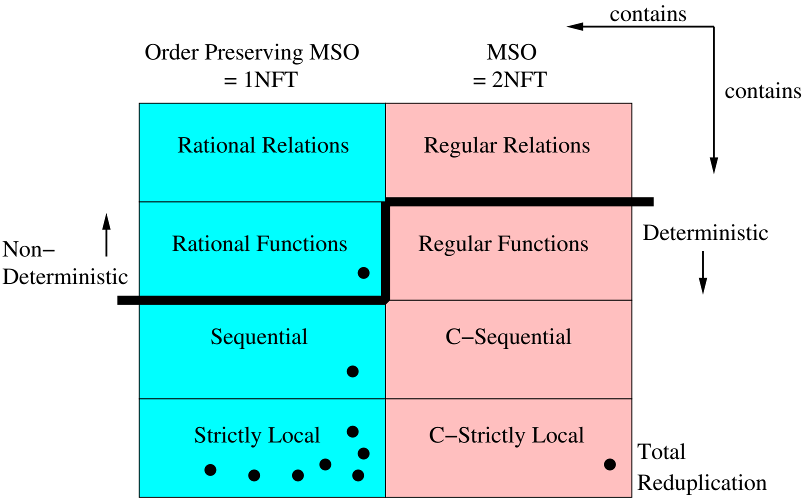
For string relations, there are!

$$\underbrace{\underbrace{[\![\text{1DFT}]\!] \subsetneq [\![\text{1fNFT}]\!] \subsetneq [\![\text{1NFT}]\!]}_{\text{Rational}} \sim [\![\text{2DFT}]\!] \subsetneq [\![\text{2NFT}]\!]}_{\text{Regular}}$$

1/2	1-way or 2-way
N/D	Non-/Deterministic
FT	Finite-state Transducer
f	functional

(Filiot and Reynier 2016)

String Relations



(Filiot and Reynier 2016, Chandlee 2017, Dolatian and Heinz 2020)

Part II

Making vs Recognizing Copies

Making copies vs. Recognizing copies

It is easier to *make* a copy than to *recognize* a copy.

1. Given w, is there a v such that $w = vv$? (recognizing copies)
2. Given w, return ww. (making copies)

The act of copying is *regular* but not *rational*.
Recognizing copies is *neither*.

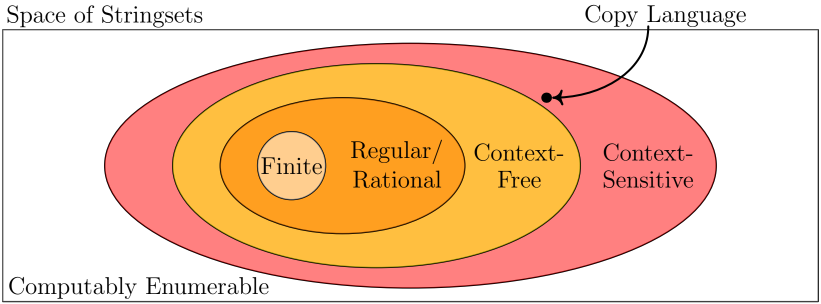
(Filiot and Reynier 2016, Dolatian and Heinz 2020)

Why Recognizing Copies is Not Regular

$$L_{\mathsf{copy}} := \{ww : w \in \Sigma^*\}$$

Three proofs

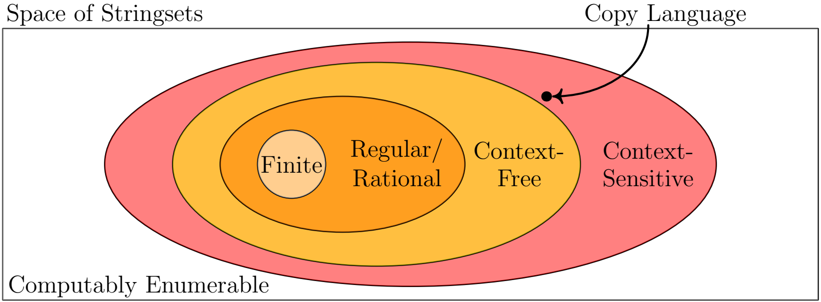
1. The Myhill/Nerode equivalence relation on L_{copy} has infinite index.
2. Proof by the Regular Pumping Lemma
3. Proof by the Context-Free Pumping Lemma



c.f. cross-serial dependencies in syntax (Shieber 1985)

Why Recognizing Copies is Not Regular

$$L_{\mathsf{copy}} := \{ww : w \in \Sigma^*\}$$



Consequently there is no way to recognize L_{copy} with any of these extensionally-equivalent formalisms:

1DFA, 1NFA, 2NFA, RE, GRE, MSO(+1), MSO(<)

Why Making Copies is not Rational

$$f_{\mathsf{copy}} := \{(w, ww) : w \in \Sigma^*\} \quad ; \qquad (\forall w \in \Sigma^*)\ f_{\mathsf{copy}}(w) := ww$$

Proof in 2 steps:

- The image of a rational relation is a rational language (Scott and Rabin 1959).
- The image of f_{copy} is L_{copy}, which is not rational.

The issue is you have to keep track of the entirety of w, which can be arbitrarily large, to know to write another w when you get to the end of the string (and you can only "read" the string once and only "write" once.)

Why Making Copies is Regular

$$f_{\mathsf{copy}} := \{(w, ww) : w \in \Sigma^*\} \quad ; \qquad (\forall w \in \Sigma^*)\ f_{\mathsf{copy}}(w) := ww$$

Proof by Construction

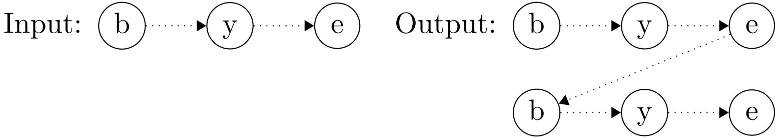
1. Logical transductions (MSO with Successor)
2. 2-way Deterministic Finite-State Transducer
3. Cost Register Automata

(Engelfriedt and Hoogeboom 2001, Alur et al. 2013)

Making Copies with Logical Transductions

$$\langle \mathcal{D}; \triangleleft, \sigma \rangle_{\sigma \in \Sigma} \quad \rightarrow \quad \langle \mathcal{D}; \triangleleft, \sigma \rangle_{\sigma \in \Sigma}$$

$$\begin{array}{rcl}
\varphi_{\texttt{domain}} & := & \texttt{true} \\
C & := & \{1, 2\} \\
\varphi^1_{\texttt{license}}(x) = \varphi^2_{\texttt{license}}(x) & := & \texttt{true} \\
\varphi^{1,1}_{\triangleleft}(x,y) = \varphi^{2,2}_{\triangleleft}(x,y) & := & x \triangleleft y \\
\varphi^{1,2}_{\triangleleft}(x,y) & := & \textsf{word-final}(x) \wedge \textsf{word-initial}(y) \\
\varphi^{2,1}_{\triangleleft}(x,y) & := & \texttt{false} \\
\forall \sigma \in \Sigma : \varphi^1_{\sigma}(x) = \varphi^2_{\sigma}(x) & := & \sigma(x)
\end{array}$$

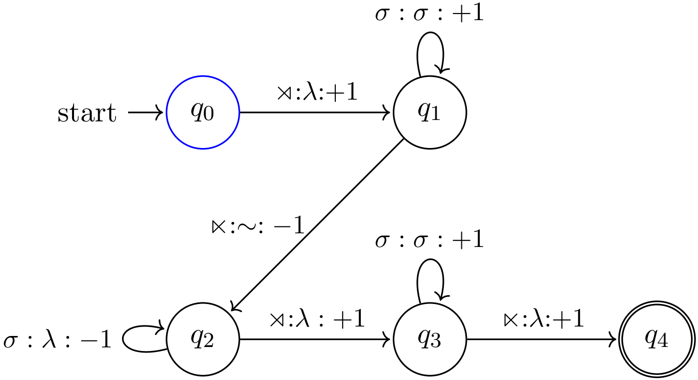


Making Copies with 2-way DFTs

- Working example: bye→bye∼bye

  Input: ⋊ b y e ⋉

  Output:

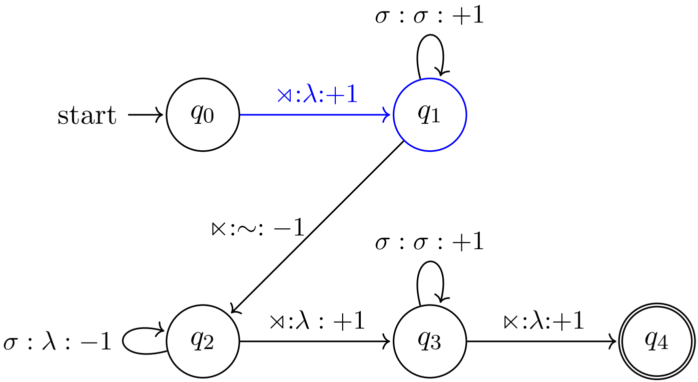


Making Copies with 2-way DFTs

- Working example: bye→bye∼bye

  Input: ⋊ b y e ⋉

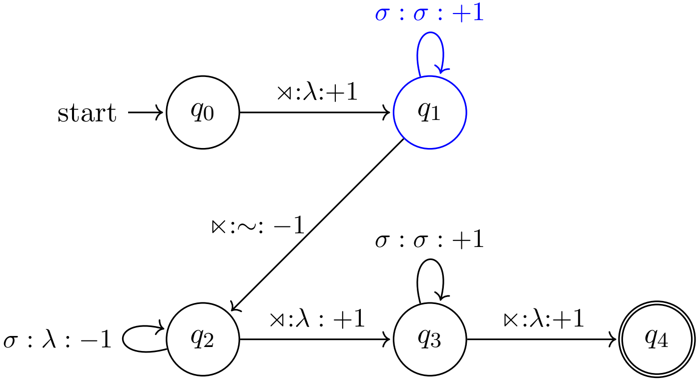
  Output:



Making Copies with 2-way DFTs

- Working example: bye→bye∼bye

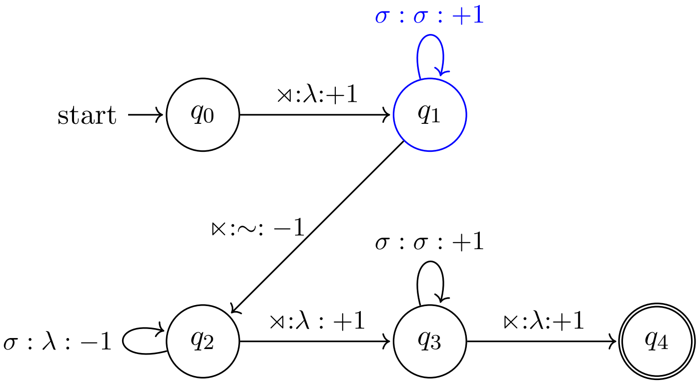
Input:	$\rtimes$	b	y	e	$\ltimes$
Output:		b			



Making Copies with 2-way DFTs

- Working example: bye→bye∼bye

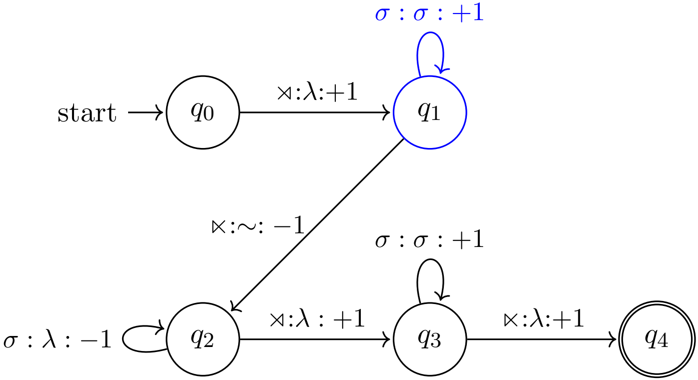
Input:	⋊	b	y	e	⋉
Output:		b	y		



Making Copies with 2-way DFTs

- Working example: bye→bye∼bye

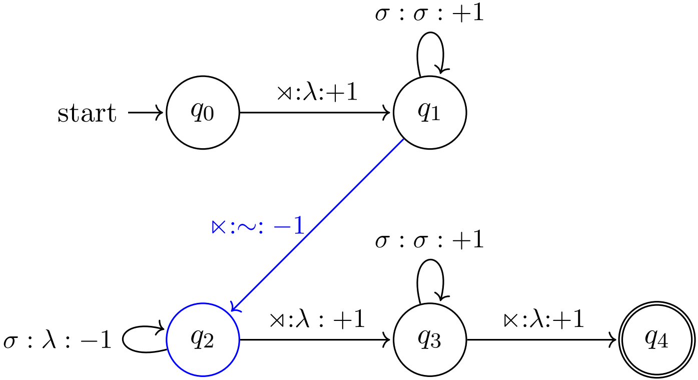
Input:	⋊	b	y	e	⋉
Output:		b	y	e	



Making Copies with 2-way DFTs

- Working example: bye→bye∼bye

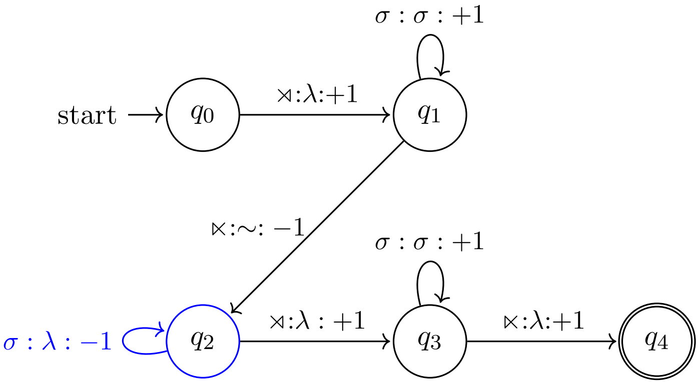
Input:	$\rtimes$	b	y	e	$\ltimes$
Output:		b	y	e	$\sim$



Making Copies with 2-way DFTs

- Working example: bye→bye∼bye

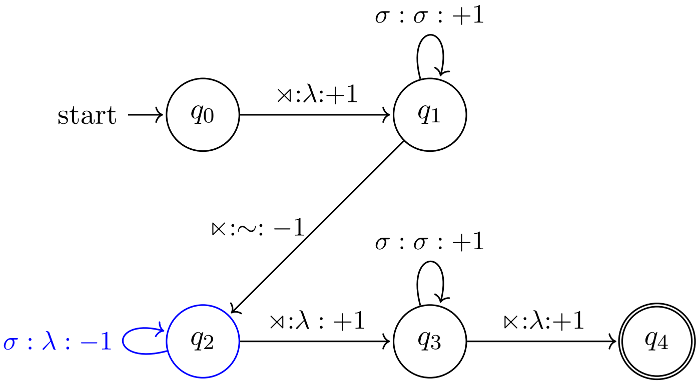
Input:	$\rtimes$	b	y	e	$\ltimes$
Output:		b	y	e	$\sim$



Making Copies with 2-way DFTs

- Working example: bye→bye∼bye

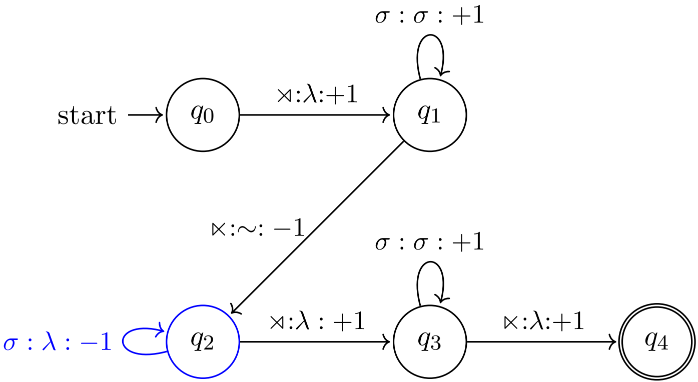
Input:	$\rtimes$	b	y	e	$\ltimes$
Output:		b	y	e	$\sim$



Making Copies with 2-way DFTs

- Working example: bye→bye∼bye

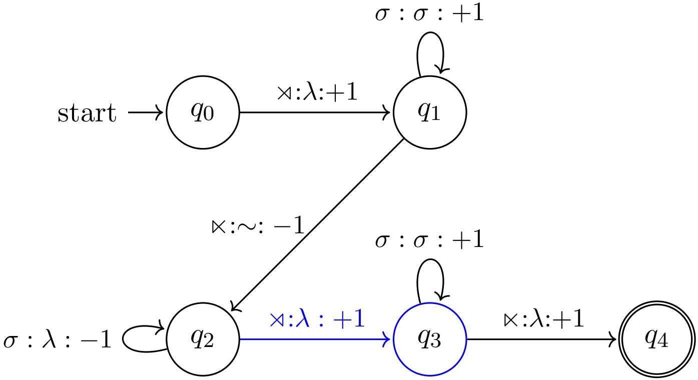
Input:	$\rtimes$	b	y	e	$\ltimes$
Output:		b	y	e	$\sim$



Making Copies with 2-way DFTs

- Working example: bye→bye∼bye

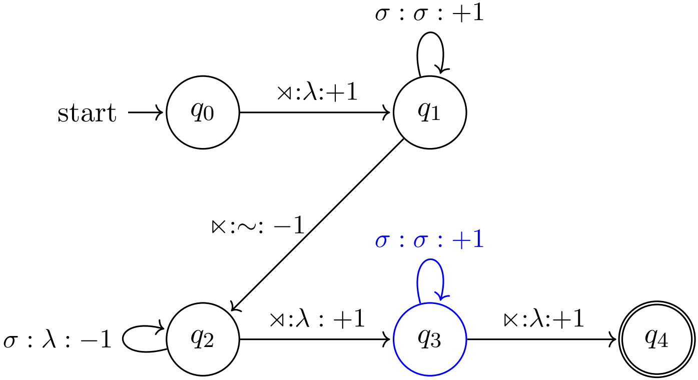
Input:	$\rtimes$	b	y	e	$\ltimes$
Output:		b	y	e	$\sim$



Making Copies with 2-way DFTs

- Working example: bye→bye∼bye

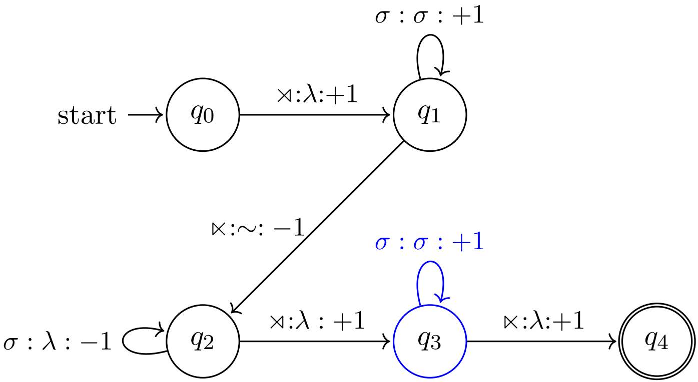
Input:	$\rtimes$	b	y	e	$\ltimes$
Output:		b	y	e	$\sim$
		b			



Making Copies with 2-way DFTs

- Working example: bye→bye∼bye

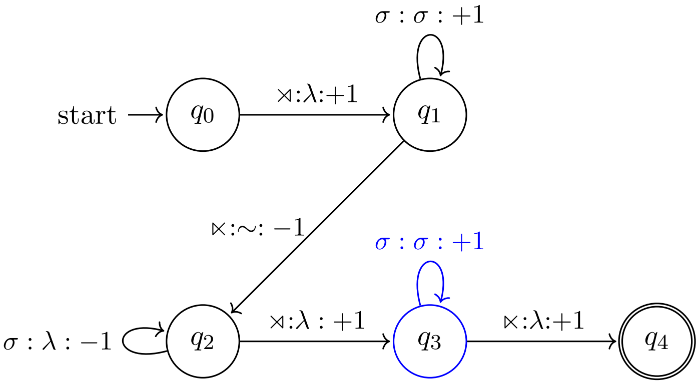
Input:	$\rtimes$	b	y	e	$\ltimes$
Output:		b	y	e	$\sim$
		b	y		



Making Copies with 2-way DFTs

- Working example: bye→bye∼bye

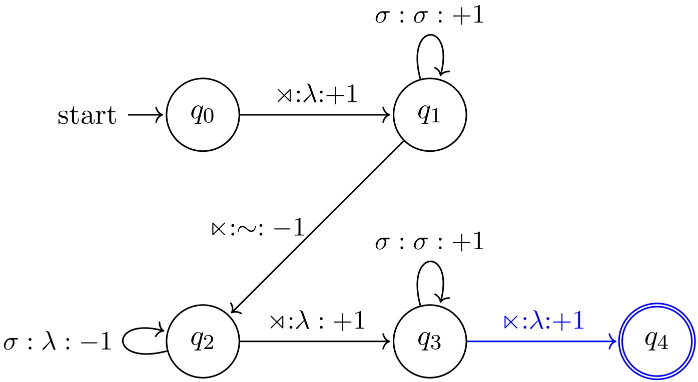
Input:	$\rtimes$	b	y	e	$\ltimes$
Output:		b	y	e	∼
		b	y	e	



Making Copies with 2-way DFTs

- Working example: bye→bye∼bye

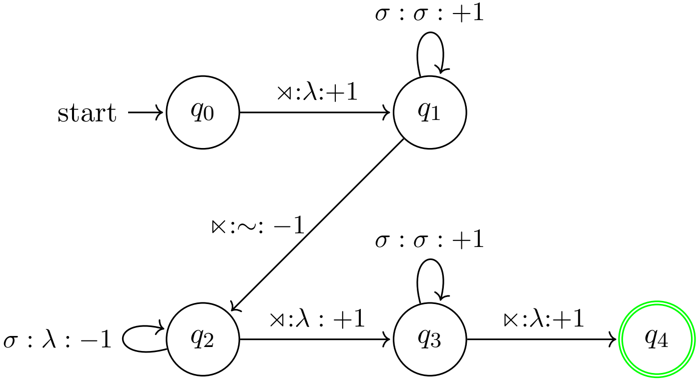
Input:	⋊	b	y	e	⋉
Output:		b	y	e	∼
		b	y	e	



Making Copies with 2-way DFTs

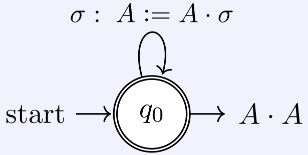
- Working example: bye→bye∼bye

Input:	$\rtimes$	b	y	e	$\ltimes$
Output:		b	y	e	$\sim$
		b	y	e	



Making Copies with Cost Register Automata

- Working example: $\rtimes$bye$\ltimes$ $\quad\rightarrow\quad$ bye$\sim$bye



register A: λ

(cf. Cohen-Sygal and Wintner 2006)

Each of these constructions concatenate 1-way-definable functions!

$$f_{\texttt{copy}}(w) = \texttt{id}(w) \cdot \texttt{id}(w)$$

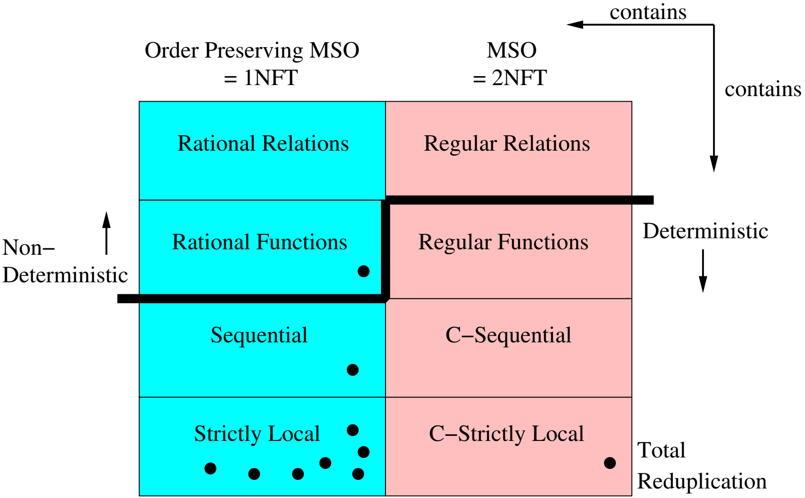
Each of these constructions concatenate 1-way-definable functions!

$$f_{\texttt{RED}}(w) = \underbrace{g(w)} \cdot \underbrace{h(w)}$$

What kinds of functions are g and h?

String Relations

A more articulated view



(Filiot and Reynier 2016, Chandlee 2017, Dolatian and Heinz 2020)

Linguistically interesting observation #1

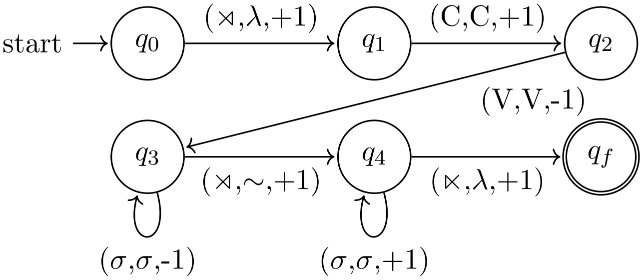
$$f_{\texttt{RED}}(w) = \underbrace{g(w)} \cdot \underbrace{h(w)}$$

1. Total reduplication is the case where $g = h =$`id`.
2. Partial reduplication can be understood where either g or h is truncation, and the other is `id`.
3. More complex cases can be understood as variations on these themes.

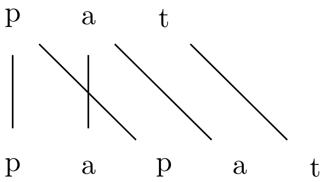
(cf. Steriade 1988)

Linguistically interesting observation #2

The semantics of regular formalisms is *copying*, not memorization.



pat $\xrightarrow{\texttt{CV-RED}}$ pa-pat



(Bojańczyk 2014)

Interim Summary

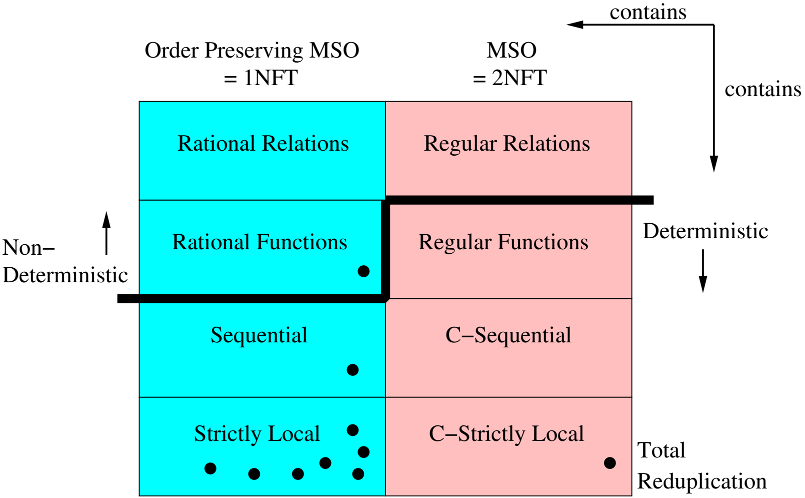
1. 'Rational' and 'Regular' mean the same thing for *stringsets* but not for *string relations*.
2. Making copies with regular formalisms such as logic, 2way DFT, and register automata are easy.
3. Recognizing copies is more complex.
4. Regular formalisms make better contact with linguistic theories of reduplication than rational formalisms.
5. Do the reduplicative patterns in the world's languages need the full power of the regular formalisms?

Interim Summary

1. 'Rational' and 'Regular' mean the same thing for *stringsets* but not for *string relations*.
2. Making copies with regular formalisms such as logic, 2way DFT, and register automata are easy.
3. Recognizing copies is more complex.
4. Regular formalisms make better contact with linguistic theories of reduplication than rational formalisms.
5. Do the reduplicative patterns in the world's languages need the full power of the regular formalisms?
6. Probably not.

String Relations

A more articulated view



(Filiot and Reynier 2016, Chandlee 2017, Dolatian and Heinz 2020)

Part III

RedTyp

Building a computationally explicit typology of patterns

`RedTyp`

- is a SQL database of reduplicative processes;
- models 138 reduplicative processes across 90 languages using 57 2DFTs;
- pulled generalizations from Moravcsik (1978), Rubino (2005), Inkelas and Downing (2015), McCarthy and Prince (1995) and others.
- Some include morpho-phonological interactions.

(Dolatian and Heinz 2019)

Technical details

- Average # of states = 8.8
- Largest 2DFT has 30 states (would be 1000s for 1 way transducer)
- Python implementation
- `http://github.com/jhdeov/RedTyp`

Utility of RedTyp

1. Comparative typology (Types)
2. Computational Models of Learning Reduplication
3. . . .

Example of Modelling Learning

- Use RedTyp to generate training, development, and test sets for machine learning experiments.

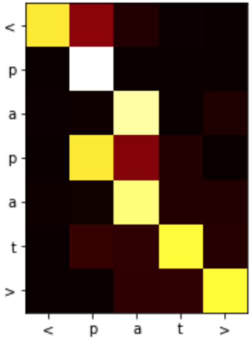
"Probing RNN Encoder-Decoder Generalization of Subregular Functions Using Reduplication" by Max Nelson, Hossep Dolatian, Jonathan Rawski, Brandon Prickett (2020)

- They show attention weights on RNNs mirror 2DFT processing, which suggests the RNNs are approximating them. (And RNNs without attention fail to learn.)

Example of Modelling Learning

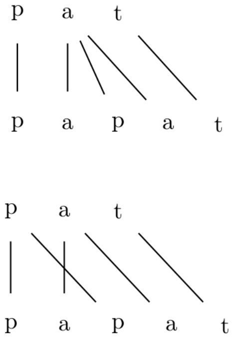
"Probing RNN Encoder-Decoder Generalization of Subregular Functions Using Reduplication" by Max Nelson, Hossep Dolatian, Jonathan Rawski, Brandon Prickett (2020)

Attention and Origin Semantics



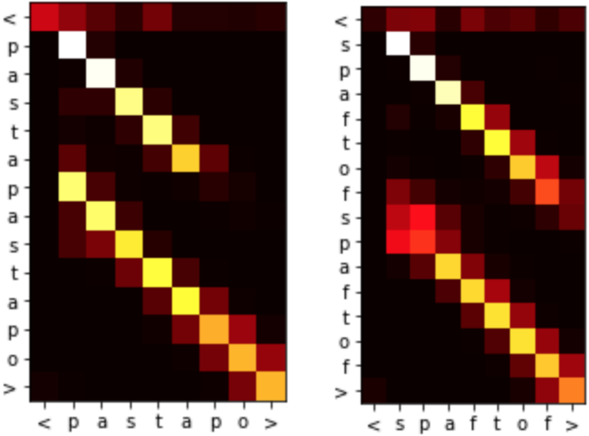
1-Way:

2-Way:



Example of Modelling Learning

"Probing RNN Encoder-Decoder Generalization of Subregular Functions Using Reduplication" by Max Nelson, Hossep Dolatian, Jonathan Rawski, Brandon Prickett (2020)



Part IV

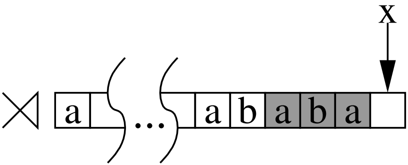
Refining the Encyclopedia of Categories

Local String-to-String Functions

- What could it mean for a string-to-string function to be local?
- Consider the Markov property.

$$P(a_{n+1} \mid a_1 a_2 \ldots a_n) \approx P(a_{n+1} \mid a_{n-k} a_{n-k+1} \ldots a_n)$$

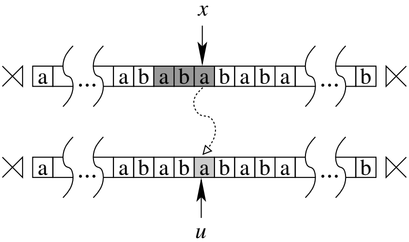
- The probability of the next item only depends on the previous k symbols.



Strictly Local Functions

Chandlee develops the same idea in the context of string rewriting.

Input Strictly Local

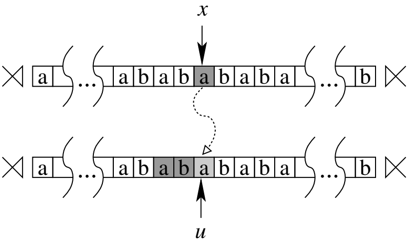


(Chandlee 2014, Chandlee et al. 2014, 2015)

Strictly Local Functions

Chandlee develops the same idea in the context of string rewriting.

Output Strictly Local

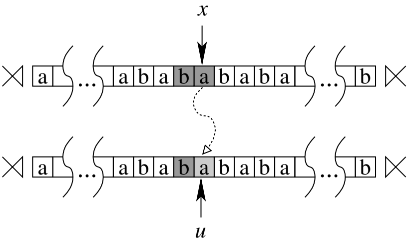


(Chandlee 2014, Chandlee et al. 2014, 2015)

Strictly Local Functions

Chandlee develops the same idea in the context of string rewriting.

Input-Output Strictly Local

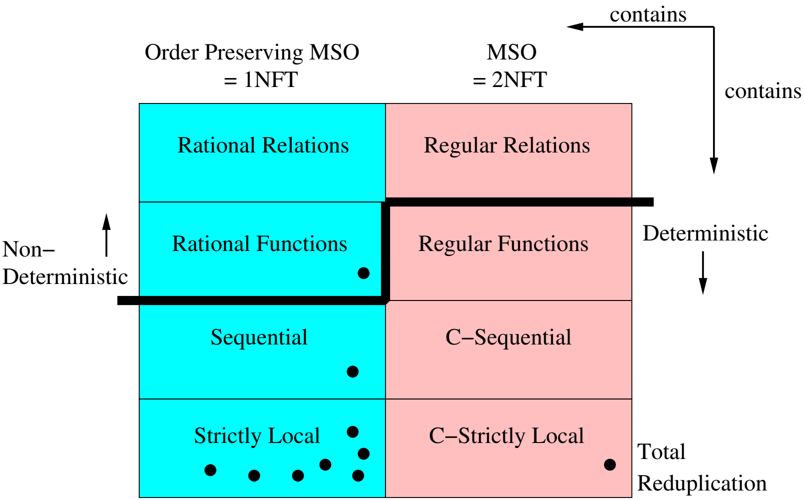


(Chandlee 2014, Chandlee et al. 2014, 2015)

Strictly Local Functions

- In the diagram, I group all of these types here as "strictly local functions."
- They cover quite a bit of the typology of morphology and phonology (Chandlee 2017, Chandlee and Heinz 2018)
- Notably they cannot capture phonological processes where the conditioning environment is unboundedly far from the target (e.g. consonant harmony).
- When parameterized by a k-window, there are algorithms that learn these functions from (input,output) pairs with good theoretical guarantees (Chandlee et al. 2014, 2015).

Strictly Local Functions



Truncation is Strictly Local

- English nicknames: truncate name to first (C)VC

/dʒɛfri/	→ [dʒɛf]	‘Jeffrey’→‘Jeff’
/deɪvɪd/	→ [deɪv]	‘David’→‘Dave’
/ælən/	→ [æl]	‘Alan→Al’

- Specifically, this example is Output Strictly Local with $k = 3$ because one needs to keep track of the last 2 segments output and the current segment. Everything after the first VC is deleted.

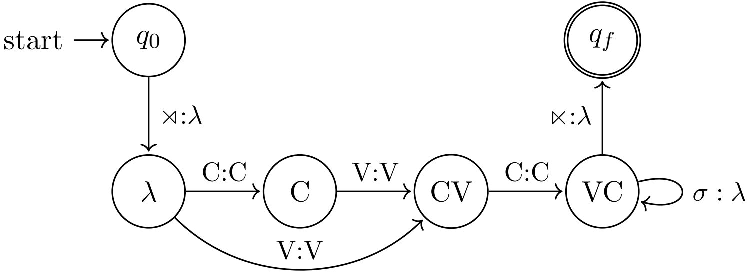
Truncation is Strictly Local

- Working example: 'Samuel' /sæmjəl/→[sæm]

Truncation is Strictly Local

- Working example: 'Samuel' /sæmjəl/→[sæm]

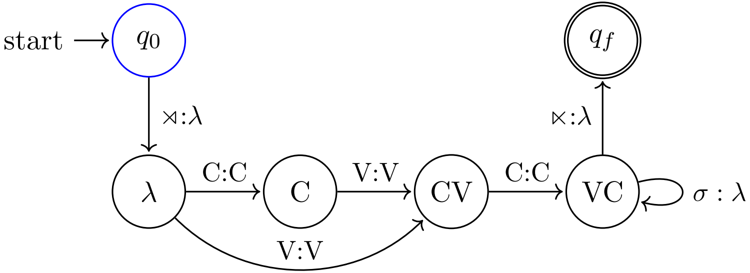
Input:	$\rtimes$	s	æ	m	j	ə	l	$\ltimes$
Output:								



Truncation is Strictly Local

- Working example: 'Samuel' /sæmjəl/→[sæm]

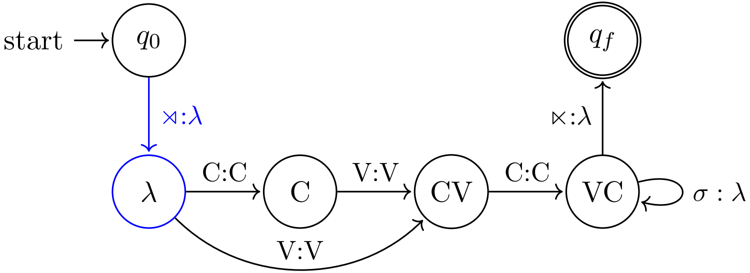
Input:	⋊	s	æ	m	j	ə	l	⋉
Output:								



Truncation is Strictly Local

- Working example: 'Samuel' /sæmjəl/→[sæm]

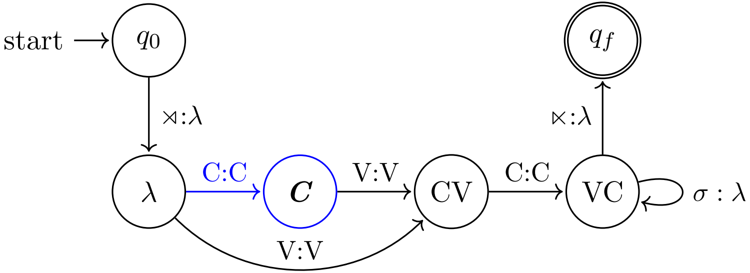
Input:	⋊	s	æ	m	j	ə	l	⋉
Output:								



Truncation is Strictly Local

- Working example: ‘Samuel’ /sæmjəl/→[sæm]

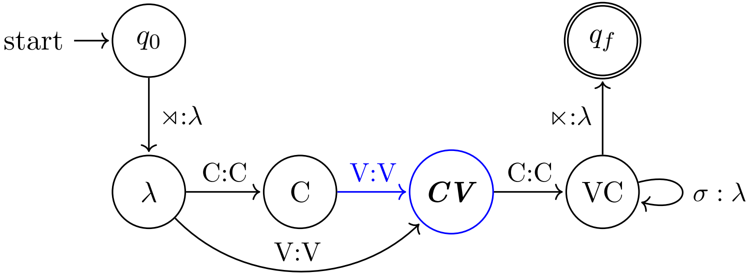
Input:	⋊	s	æ	m	j	ə	l	⋉
Output:		s						



Truncation is Strictly Local

- Working example: ‘Samuel’ /sæmjəl/→[sæm]

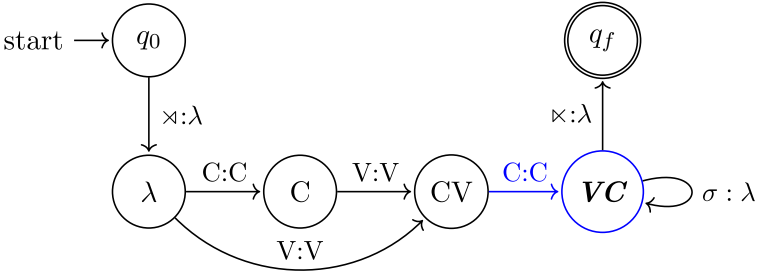
Input:	⋊	s	æ	m	j	ə	l	⋉
Output:		s	æ					



Truncation is Strictly Local

- Working example: ‘Samuel’ /sæmjəl/→[sæm]

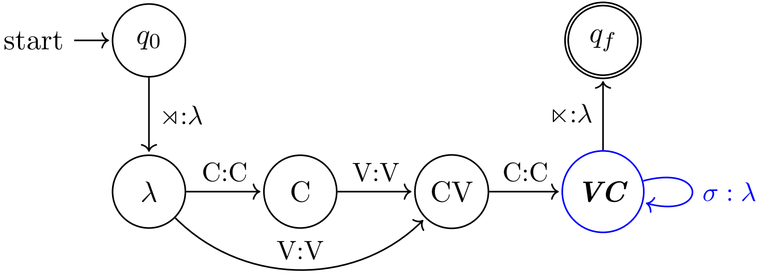
Input:	⋊	s	æ	m	j	ə	l	⋉
Output:		s	æ	m				



Truncation is Strictly Local

- Working example: ‘Samuel’ /sæmjəl/→[sæm]

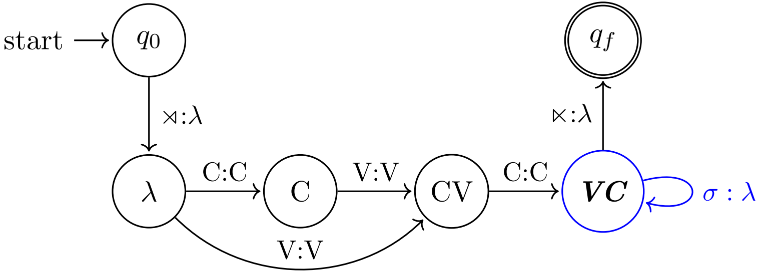
Input:	⋊	s	æ	m	j	ə	l	⋉
Output:		s	æ	m				



Truncation is Strictly Local

- Working example: ‘Samuel’ /sæmjəl/→[sæm]

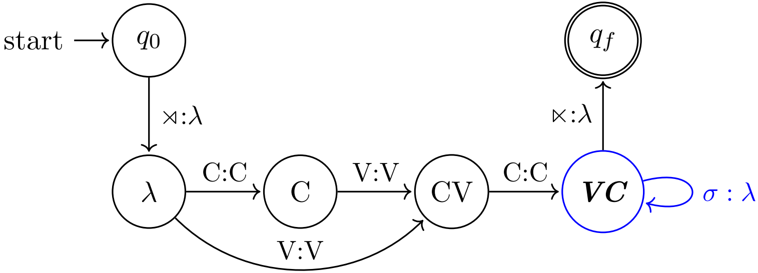
Input:	$\rtimes$	s	æ	m	j	ə	l	$\ltimes$
Output:		s	æ	m				



Truncation is Strictly Local

- Working example: 'Samuel' /sæmjəl/→[sæm]

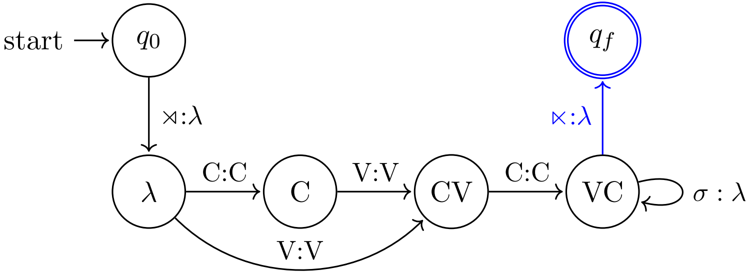
Input:	⋊	s	æ	m	j	ə	l	⋉
Output:		s	æ	m				



Truncation is Strictly Local

- Working example: ‘Samuel’ /sæmjəl/→[sæm]

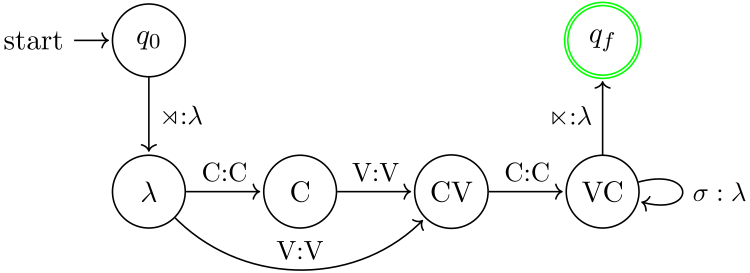
Input:	⋊	s	æ	m	j	ə	l	⋉
Output:		s	æ	m				



Truncation is Strictly Local

- Working example: ‘Samuel’ /sæmjəl/→[sæm]

Input:	⋊	s	æ	m	j	ə	l	⋉
Output:		s	æ	m				



Part V

Comparing the Types with the Categories

How much of RedTyp is C-Strictly Local?

- 87% of the 57 distinct reduplicative patterns in RedTyp are the Concatenation of Output Strictly Local functions (C-OSL).
- This includes total reduplication.
- The other ∼13% are
  1. Concatenation of *Sequential* functions.
  2. Potentially require *Compositions* of Strictly Local or Sequential functions.
  3. These deserve further study.

Learning C-OSL

- Learning k-C-OSL can be reduced to learning the 1-way k-OSL functions (Dolatian and Heinz 2018).
    1. If the boundary is overtly represented, then this is straightforward.
        1. Break up input data D into two data sets $D1 = \{(w, u) \mid (w, u \sim v) \in D\}$ and $D2 = \{(w, v) \mid (w, u \sim v) \in D\}$
        2. Submit D1 and D2 and k to OSLFIA (Chandlee et al. 2015) which learns 1-way transducers T1 and T2 representing OSL functions.
        3. Concatenate T1 and T2.
    2. If the boundary is latent then the learning problem reduces to finding this boundary (still an open problem).

Part VI

Conclusion

Conclusion

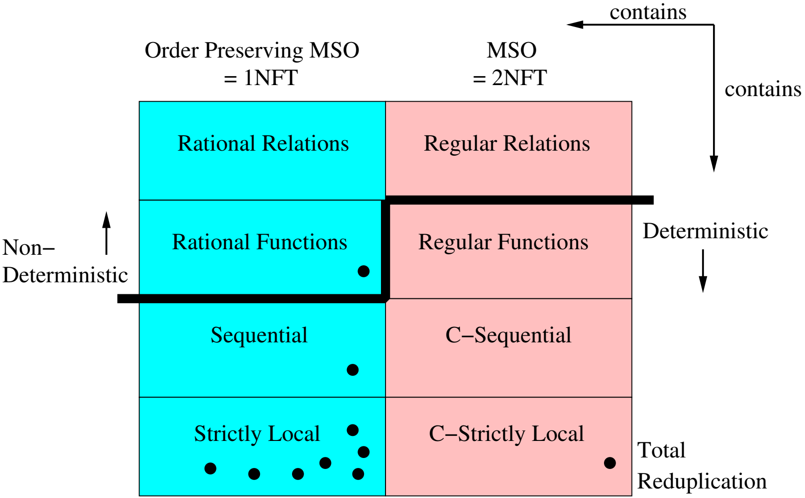
1. An encyclopedia of categories for string-to-string functions is valuable.
2. Regular functions and rational functions are different.
3. Regular formalisms make better contact with linguistic theories of reduplication than rational formalisms.
4. Strictly Local functions are a natural way to understand locality in string transformations.
5. Partial Reduplication and Total reduplication are not the outliers they appeared to be – they are mostly the concatenation of Strictly Local functions.
6. The clustering of attested types in this region also helps us understand how such patterns may be learned.

Conclusion

1. An encyclopedia of categories for string-to-string functions is valuable.
2. Regular functions and rational functions are different.
3. Regular formalisms make better contact with linguistic theories of reduplication than rational formalisms.
4. Strictly Local functions are a natural way to understand locality in string transformations.
5. Partial Reduplication and Total reduplication are not the outliers they appeared to be – they are mostly the concatenation of Strictly Local functions.
6. The clustering of attested types in this region also helps us understand how such patterns may be learned.
7. Everything that was done here with strings we can do with *trees* (Stabler 2019), and other linguistic structures using model theory and logic.

Thanks!

A more articulated view



(Filiot and Reynier 2016, Chandlee 2017, Dolatian and Heinz 2020)