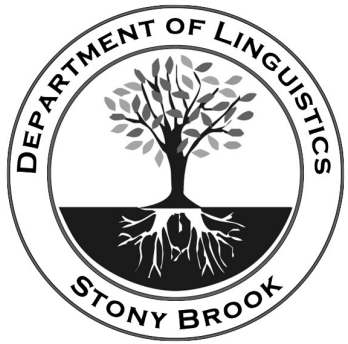
Deterministic Analyses of Optional Processes

Jeffrey Heinz



Stony Brook University

UC Irvine
December 08, 2020

Part I

The so-called Subregular Program

Doing Linguistic Typology

Requires two books:

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

Wilhelm Von Humboldt

Encyclopedia of Types

Phonological Transformations

1. Locally-triggered Assimilations
2. Locally-triggered Dissimilations
3. Syllable Structure Repairs
4. Metathesis
5. Vowel harmony
6. Consonant harmony
7. Consonant disharmony
8. Tonal Processes
9. ...

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. ...

Encyclopedia of Categories?

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

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. ...

String-to-String Functions

The established, foundational view (Roark and Sproat 2007)

Rational Relations	non-Rational Relations
All phonological processes	Total Reduplication
Affixation	
Truncation	
Root and pattern	
Umlaut/Ablaut	
Partial Reduplication	
...	

String-to-String Functions

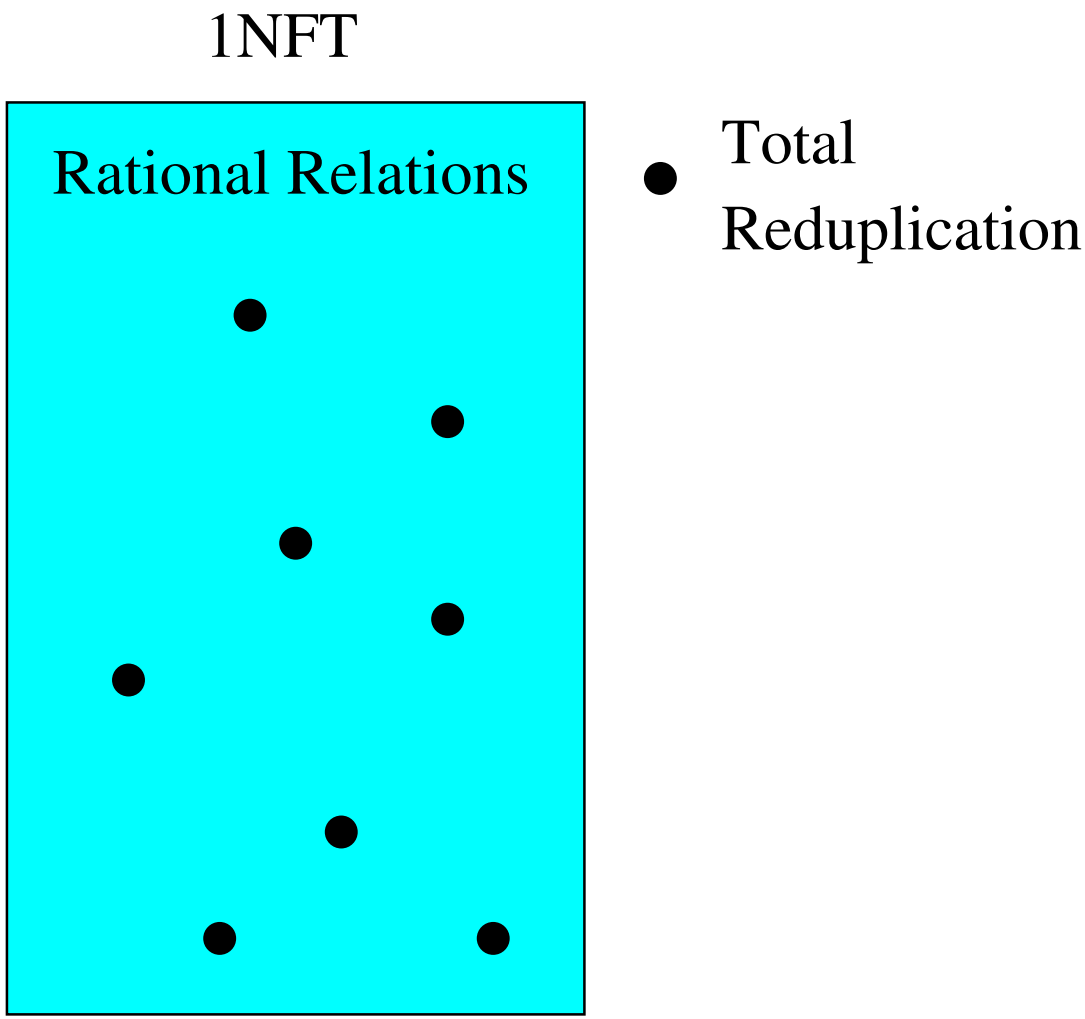
The established, foundational view (Roark and Sproat 2007)

Rational Relations	non-Rational Relations
All phonological processes	Total Reduplication
Affixation	
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.

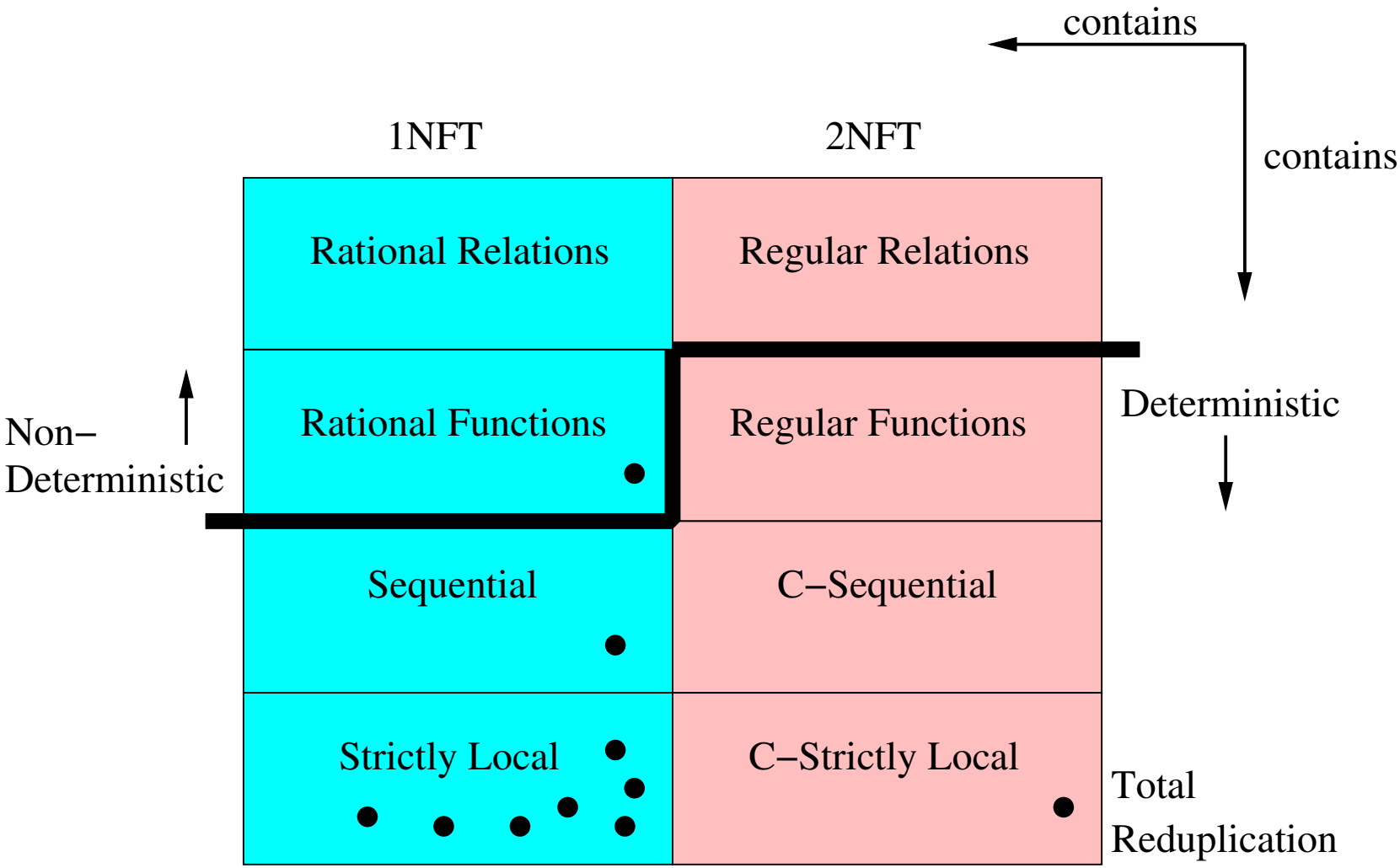
String-to-String Functions

This basic view pictorially



String-to-String Functions

A more articulated view



(Chandlee 2017, Dolatian and Heinz 2020)

Rational vs. Regular

For stringsets (formal languages) there is no distinction.

$$[\![\text{1DFA}]\!] = [\![\text{1NFA}]\!] = [\![\text{2NFA}]\!]$$

$$= [\![\text{RE}]\!] = [\![\text{GRE}]\!]$$

$$= [\![\text{MSO}(+1)]\!] = [\![\text{MSO}(<)]\!]$$

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-to-string functions, there are!

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

Part II

Is Phonology Deterministic?

Deterministic transformations in phonology

To what extent are transformations in phonology deterministic?

1. Vowel harmony (Gainor et al. 2012, Heinz and Lai 2013)
2. Metathesis (Chandlee and Heinz 2012)
3. Locally-triggered processes (Chandlee 2014, Chandlee and Heinz 2018)
4. Consonant harmony (Luo 2017)
5. Consonant disharmony (Payne 2017)
6. Unbounded Tone Plateauing (Jardine 2016)

Deterministic transformations in phonology

To what extent are transformations in phonology deterministic?

1. ✓ Vowel harmony (Gainor et al. 2012, Heinz and Lai 2013)
2. Metathesis (Chandlee and Heinz 2012)
3. Locally-triggered processes (Chandlee 2014, Chandlee and Heinz 2018)
4. Consonant harmony (Luo 2017)
5. Consonant disharmony (Payne 2017)
6. Unbounded Tone Plateauing (Jardine 2016)

Deterministic transformations in phonology

To what extent are transformations in phonology deterministic?

1. ✓ Vowel harmony (Gainor et al. 2012, Heinz and Lai 2013)
2. ✓ Metathesis (Chandlee and Heinz 2012)
3. Locally-triggered processes (Chandlee 2014, Chandlee and Heinz 2018)
4. Consonant harmony (Luo 2017)
5. Consonant disharmony (Payne 2017)
6. Unbounded Tone Plateauing (Jardine 2016)

Deterministic transformations in phonology

To what extent are transformations in phonology deterministic?

1. ✓ Vowel harmony (Gainor et al. 2012, Heinz and Lai 2013)
2. ✓ Metathesis (Chandlee and Heinz 2012)
3. ✓ Locally-triggered processes (Chandlee 2014, Chandlee and Heinz 2018)
4. Consonant harmony (Luo 2017)
5. Consonant disharmony (Payne 2017)
6. Unbounded Tone Plateauing (Jardine 2016)

Deterministic transformations in phonology

To what extent are transformations in phonology deterministic?

1. ✓ Vowel harmony (Gainor et al. 2012, Heinz and Lai 2013)
2. ✓ Metathesis (Chandlee and Heinz 2012)
3. ✓ Locally-triggered processes (Chandlee 2014, Chandlee and Heinz 2018)
4. ✓ Consonant harmony (Luo 2017)
5. Consonant disharmony (Payne 2017)
6. Unbounded Tone Plateauing (Jardine 2016)

Deterministic transformations in phonology

To what extent are transformations in phonology deterministic?

1. ✓ Vowel harmony (Gainor et al. 2012, Heinz and Lai 2013)
2. ✓ Metathesis (Chandlee and Heinz 2012)
3. ✓ Locally-triggered processes (Chandlee 2014, Chandlee and Heinz 2018)
4. ✓ Consonant harmony (Luo 2017)
5. ✓ Consonant disharmony (Payne 2017)
6. Unbounded Tone Plateauing (Jardine 2016)

Deterministic transformations in phonology

To what extent are transformations in phonology deterministic?

1. ✓ Vowel harmony (Gainor et al. 2012, Heinz and Lai 2013)
2. ✓ Metathesis (Chandlee and Heinz 2012)
3. ✓ Locally-triggered processes (Chandlee 2014, Chandlee and Heinz 2018)
4. ✓ Consonant harmony (Luo 2017)
5. ✓ Consonant disharmony (Payne 2017)
6. ✗ Unbounded Tone Plateauing (Jardine 2016)

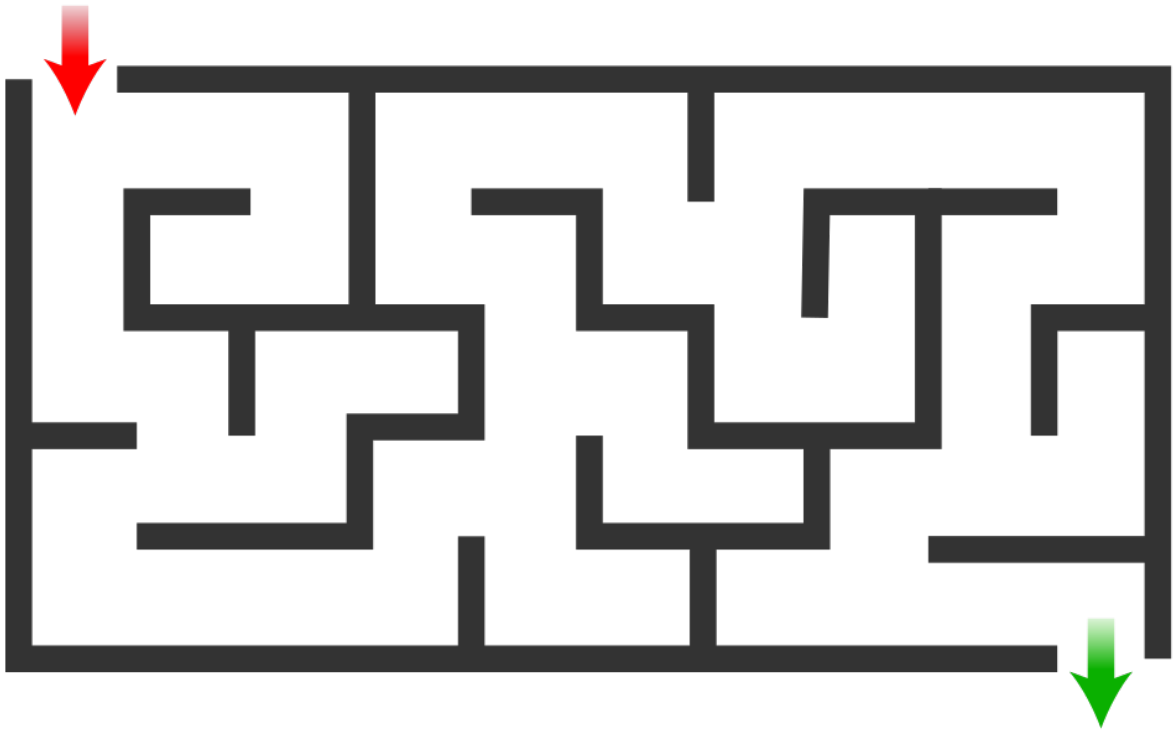
Deterministic transformations in phonology

To what extent are transformations in phonology deterministic?

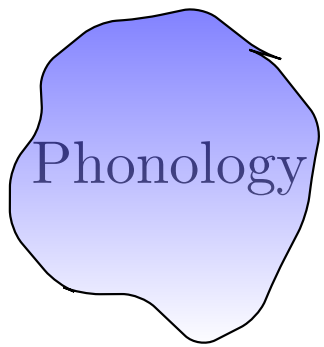
1. ✓ Vowel harmony (Gainor et al. 2012, Heinz and Lai 2013)
2. ✓ Metathesis (Chandlee and Heinz 2012)
3. ✓ Locally-triggered processes (Chandlee 2014, Chandlee and Heinz 2018)
4. ✓ Consonant harmony (Luo 2017)
5. ✓ Consonant disharmony (Payne 2017)
6. ✗ Unbounded Tone Plateauing (Jardine 2016)
7. ✗ Vowel harmony (McCollum et al. 2020)

What is 'determinism'? Why does it matter?

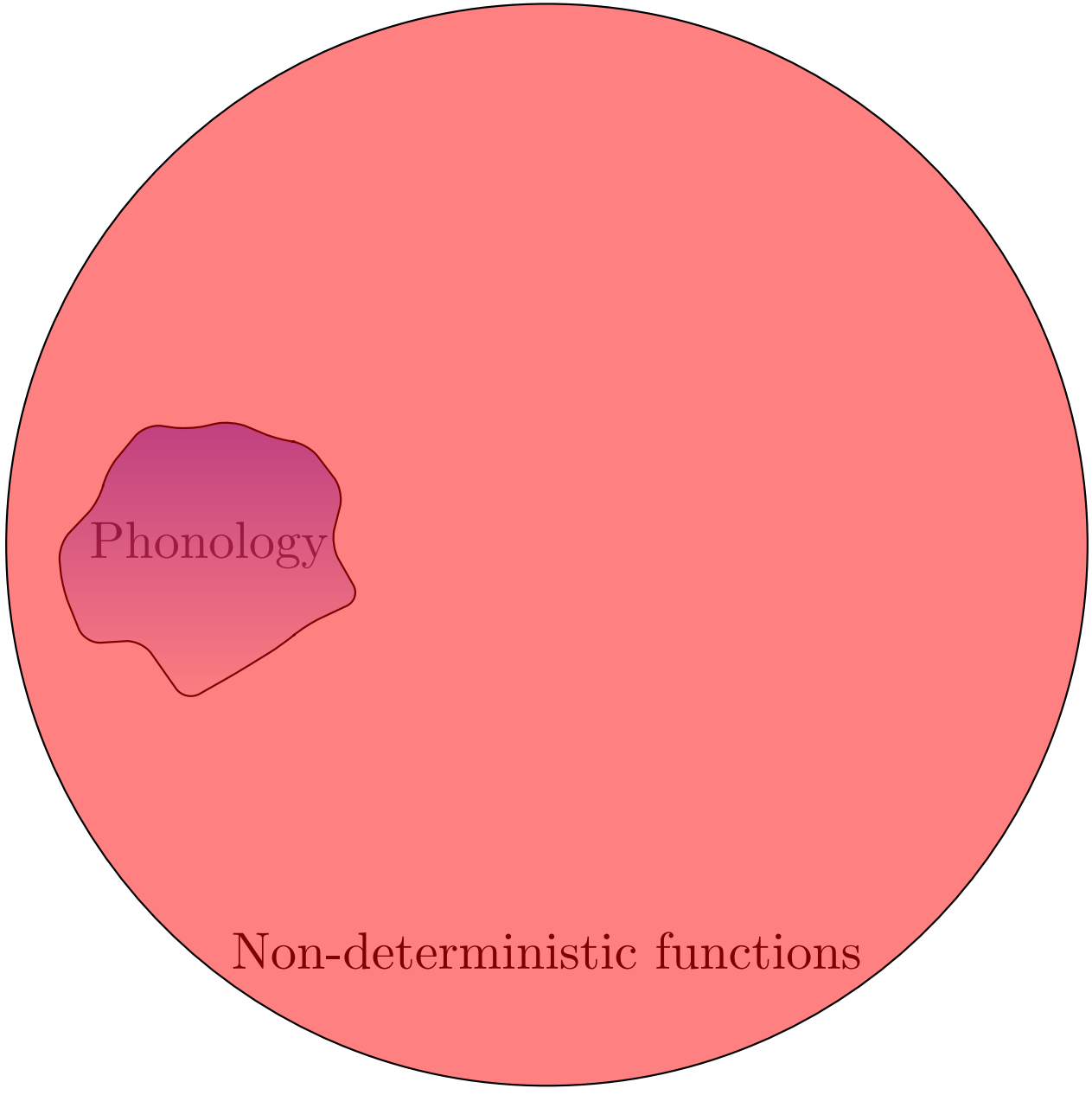
- A function f is **deterministic** iff there is an **algorithm** computing f whose execution at any time step is uniquely determined.
- It is **non-deterministic** iff there is no such algorithm—i.e. every algorithm computing f necessarily includes some time-step on some input where there is **more than one** possible path the computation can follow.



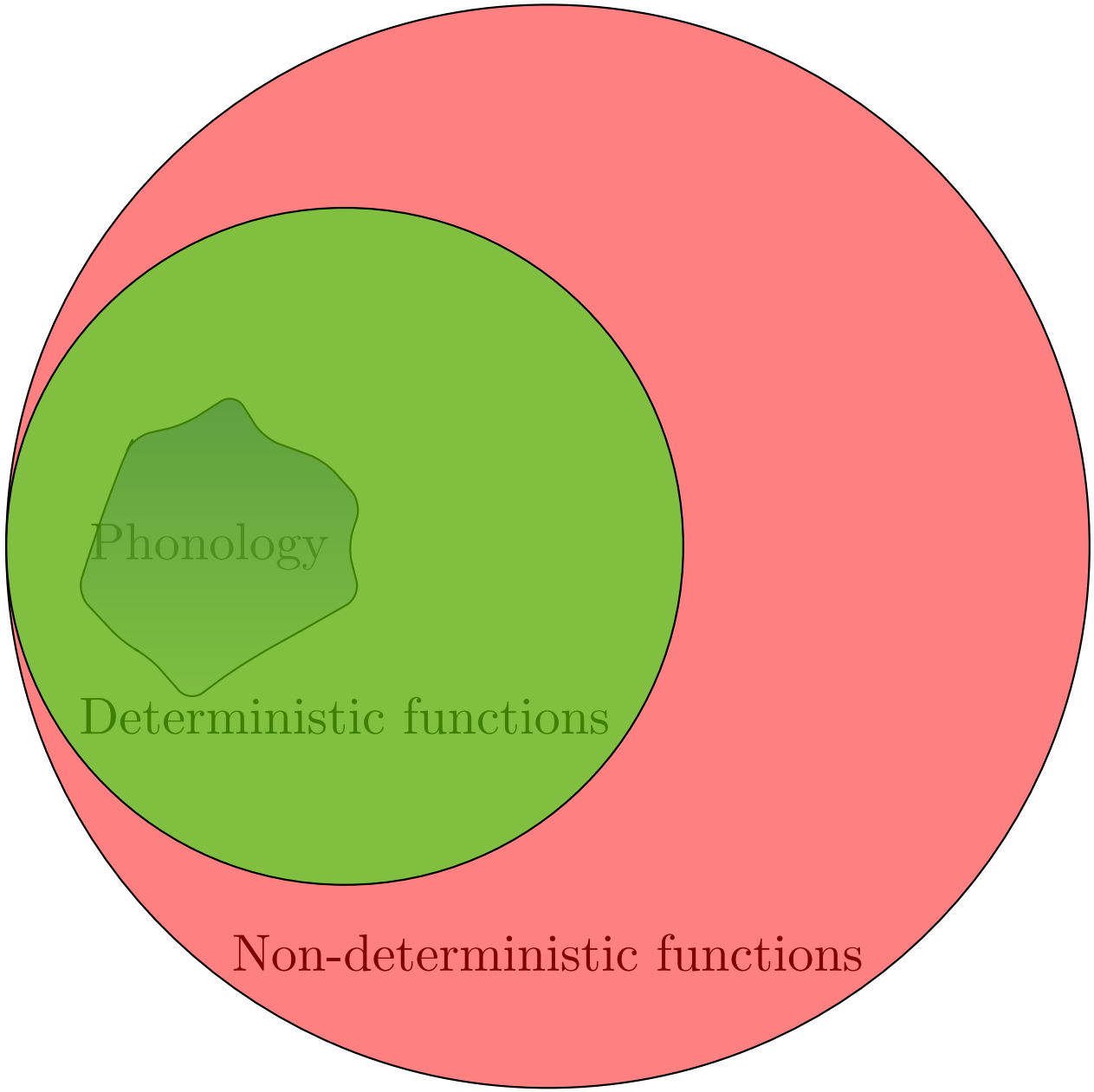
Why does it matter?



Why does it matter?



Why does it matter?



Why does it matter?

- If the hypothesis is correct, it provides a better, tighter characterization.
- We are closer to a *minimally* necessary characterization.
- A deterministic characterization **helps** learning.
  1. Smaller, better hypothesis space means there are 'fewer' hypotheses to consider.
  2. Determinism helps avoid **the credit/hidden structure problem** (Dresher and Kaye 1990, Tesar and Smolensky 2000, Heinz et al. 2015, Jarosz 2019).
- Practical: Deterministic finite-state automata process inputs in linear time, have efficient minimization algorithms, often have canonical forms for deciding equivalence and so on.

Another challenge

One challenge to the idea that phonological processes are deterministic comes from **optionality**.

McCollum et al. 2020

"...patterns of optionality like those listed in Vaux (2008) and others like iterative optionality in Icelandic umlaut (Anderson 1974) present evidence against any strong claim that segmental phonology is categorically subregular."

Today

1. I will show how iterative optionality can be **expressed and learned** with *deterministic* ISL functions building on Jardine et al. (2014) and Beros and de la Higuera (2016).
2. It will be important to rely on *phonotactic* generalizations to manage output-oriented aspects of these patterns.
3. The grammatical analysis obtained closely resembles the original proposal by Kisseberth (1970) and others.

Joint work with Kiran Eiden

Part III

Optionality, Determinism, and Strict Locality

Iterative Optionality

Vaux 2008, p. 43

(14) French schwa deletion

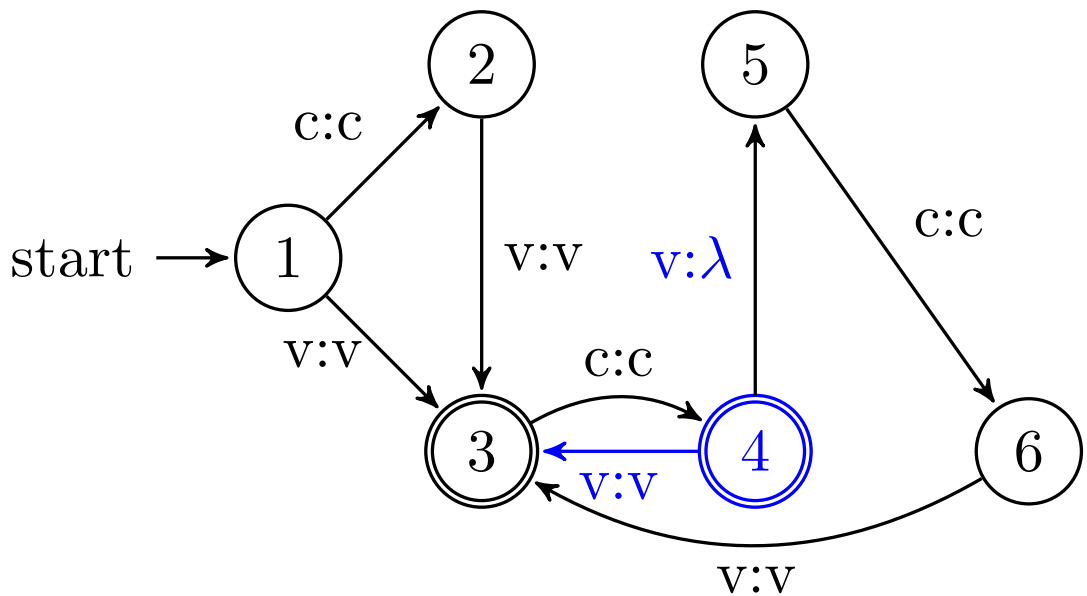
a. ə→ Ø / V (#) C _, L→R , optional across #

b. envie de te le demander 'feel like asking you' (Dell 1980: 225)

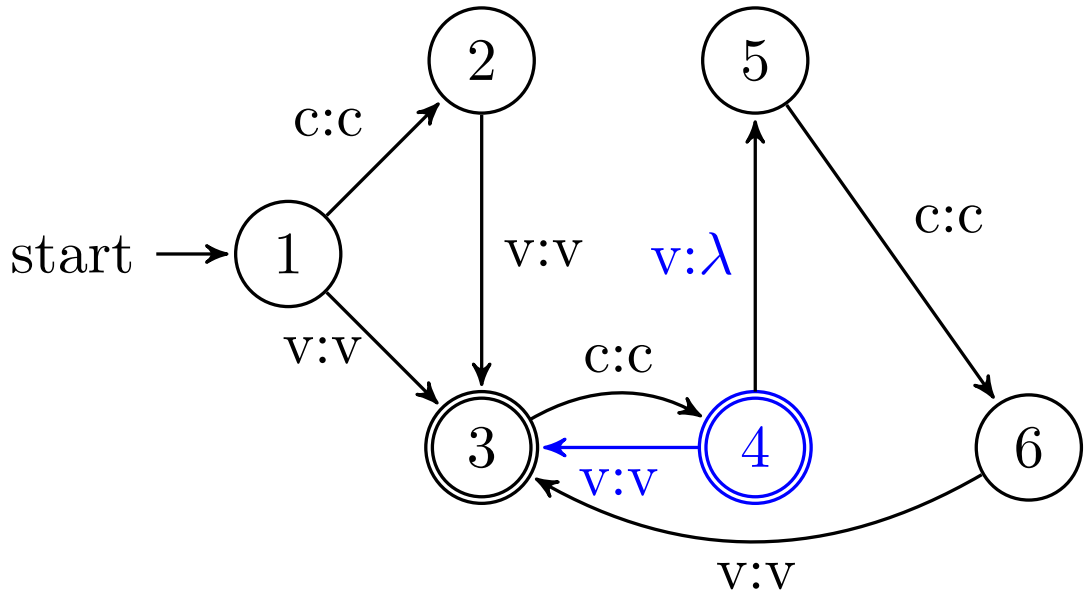
ãvidtəldəmãde
ãvidtələdəmãde
ãvidtələdmãde
ãvidətələdmãde
ãvidətlədmãde
ãvidətlədəmãde
ãvidətəldəmãde
ãvidətələdəmãde

Optional Syncope with a non-deterministic finite-state function

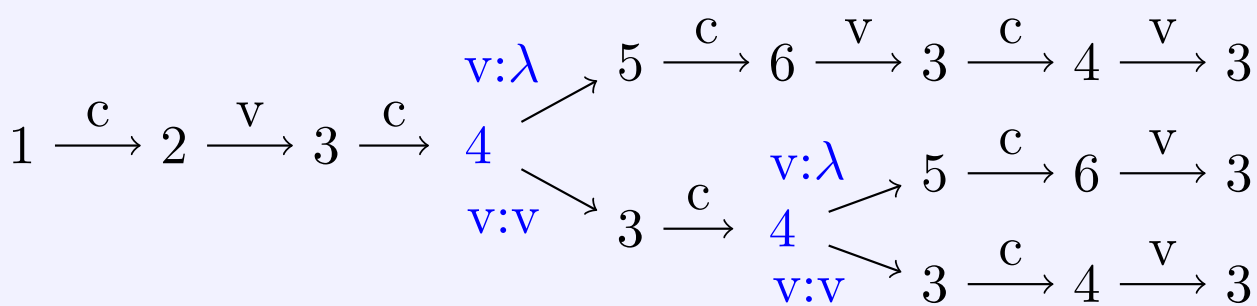
$V \rightarrow \varnothing$ / VC __ CV (applying left-to-right)



Optional Syncope with a non-deterministic finite-state function



/ c v c v c v c v /



Multiple outputs implies non-determinism, right?

- A function is **single-valued** if there is at most one output for each input.

Multiple outputs implies non-determinism, right?

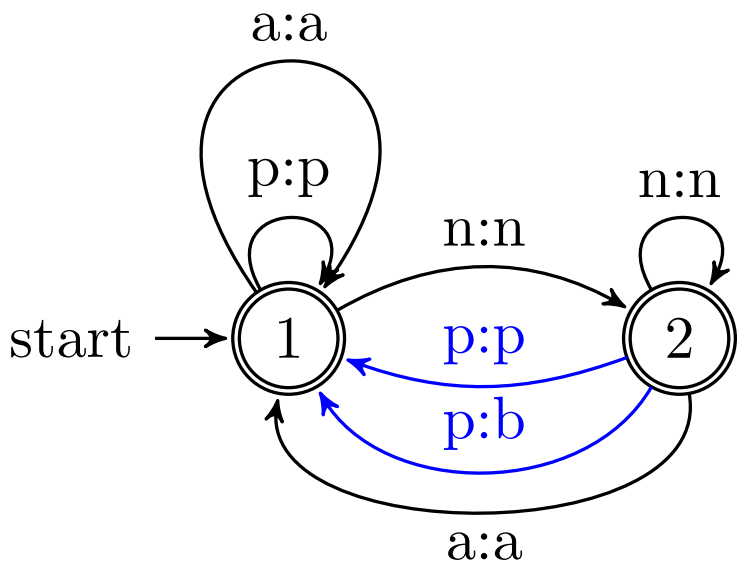
- A function is **single-valued** if there is at most one output for each input.
- What is the relationship between single-valuedness and determinism?
  1. Does single-valuedness imply determinism?
  2. Does determinism imply single-valuedness?

Multiple outputs implies non-determinism, right?

- A function is **single-valued** if there is at most one output for each input.
- What is the relationship between single-valuedness and determinism?
  1. Does single-valuedness imply determinism?
  2. Does determinism imply single-valuedness?
- I argue the answer to both questions is No.
  1. Sour-grapes Vowel Harmony is single-valued but non-deterministic (Heinz and Lai 2013).
  2. The second is more interesting; let me explain...

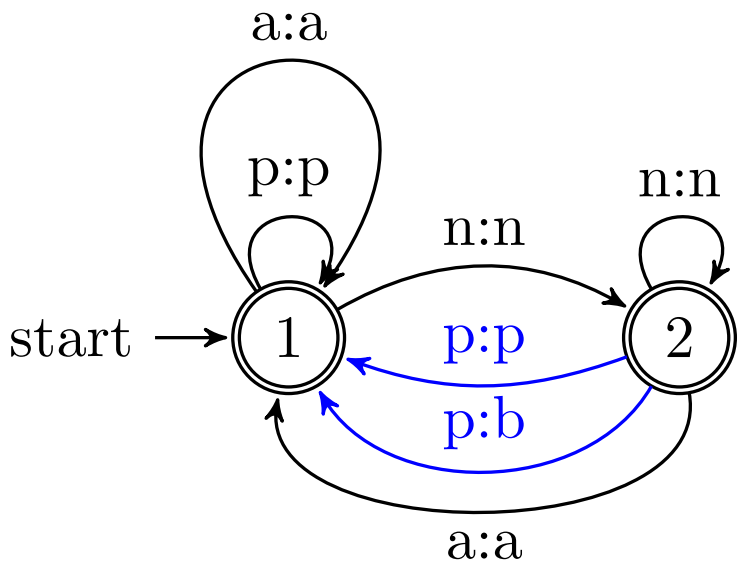
Deterministic FSTs with Language Monoids

Optional Post-nasal Voicing
(Non-deterministic)



Deterministic FSTs with Language Monoids

Optional Post-nasal Voicing
(Non-deterministic)

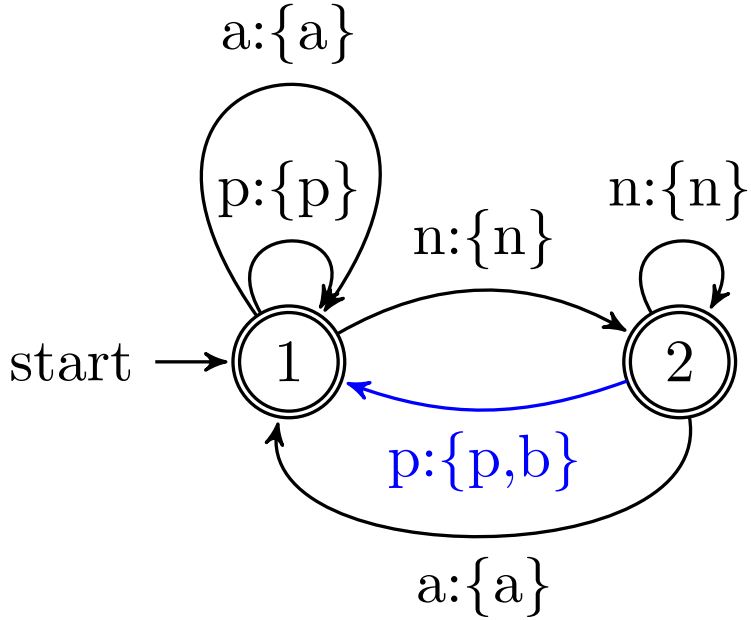


/ a n p a /

$$1 \xrightarrow{a} 1 \xrightarrow{n} \begin{matrix} \xrightarrow{p:b} 1 \xrightarrow{a} 1 \\ 2 \\ \xrightarrow{p:p} 1 \xrightarrow{a} 1 \end{matrix}$$

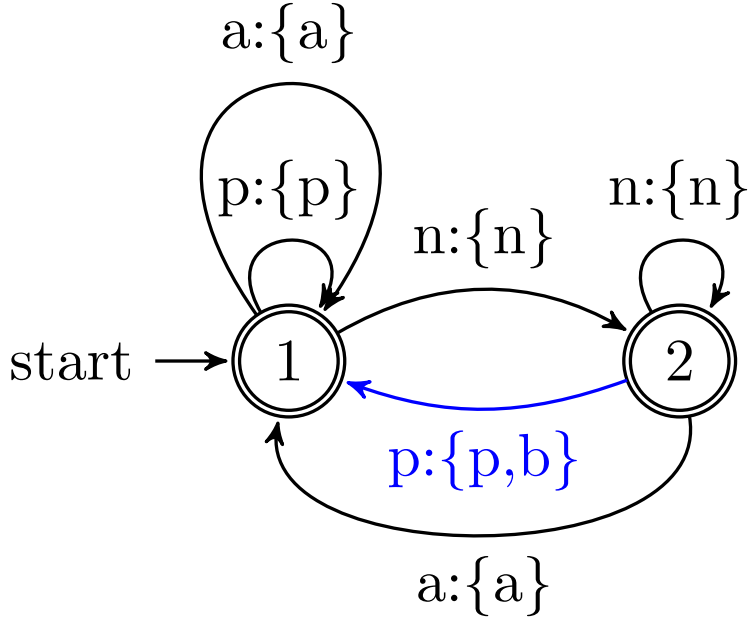
Deterministic FSTs with Language Monoids

Optional Post-nasal Voicing (Deterministic)



Beros and de la Higuera (2016) call this 'semi-determinism'.

Optional Post-nasal Voicing (Deterministic)

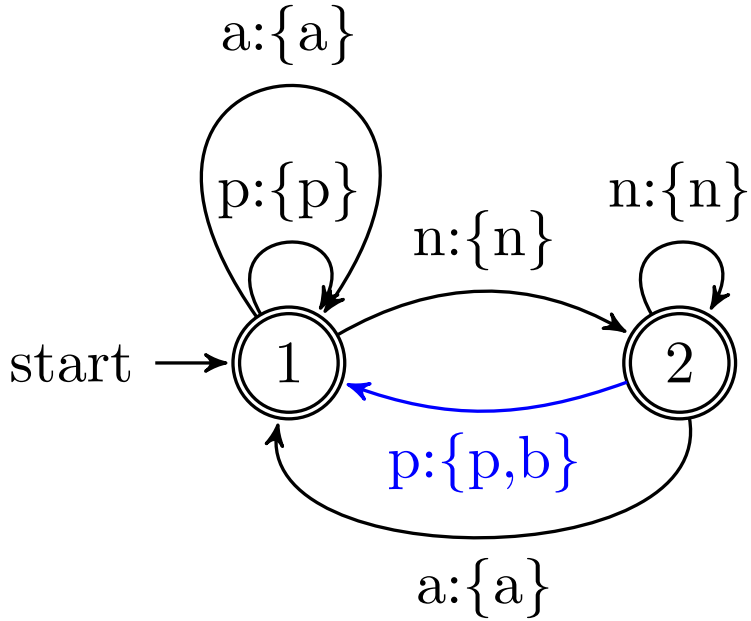


/ a n p a /

$$1 \xrightarrow{\{a\}} 1 \xrightarrow{\{n\}} 2 \xrightarrow{\{p,b\}} 1 \xrightarrow{\{a\}} 1$$

Deterministic FSTs with Language Monoids

Optional Post-nasal Voicing (Deterministic)



$$/\ a\ n\ p\ a\ / \mapsto \{a\} \cdot \{n\} \cdot \{p, b\} \cdot \{a\} = \{anpa, anba\}$$

That's the basic idea.

Monoids for Transducers

Name	K	$\otimes$	1	
String	Σ^*	$\cdot$	λ	$\Sigma^* \to \Sigma^*$
Boolean	$\{T, F\}$	$\wedge$	`true`	$\Sigma^* \to \{T, F\}$
Natural	$\mathbb{N}$	$+$	0	$\Sigma^* \to \mathbb{N}$
Real Interval	$[0, 1]$	$\times$	1	$\Sigma^* \to [0, 1]$
FIN	$\{L \subseteq \Sigma^* \mid L \text{ finite}\}$	$\cdot$	$\{\lambda\}$	$\Sigma^* \to \text{FIN}$

- Beros and de la Higuera's 'semi-determinism' is a **deterministic** string transucer whose output is drawn from the monoid of finite languages with multiplication as language concatenation (and other conditions, TBA).

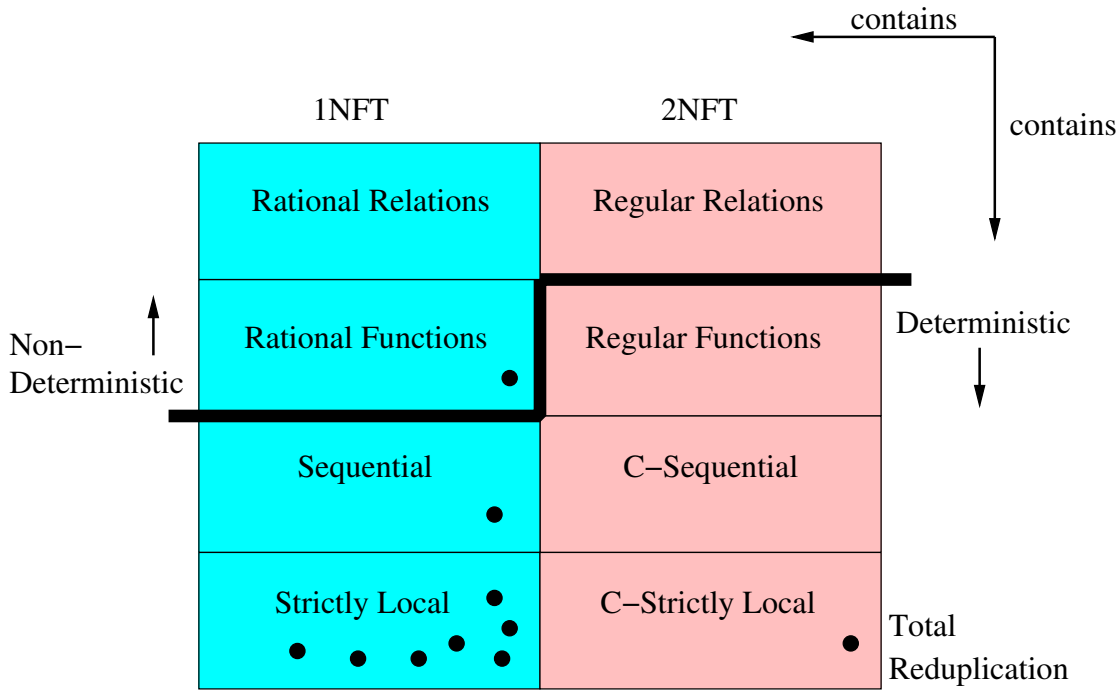
$$X \cdot Y = \{xy \mid x \in X, y \in Y\}$$

Outline of a Solution

The Input and Output Strictly local string-to-string functions

1. are deterministic
2. have properties amenable to learnability

So let's "lift" those results to transducers where the outputs are finite sets of strings to handle optionality.

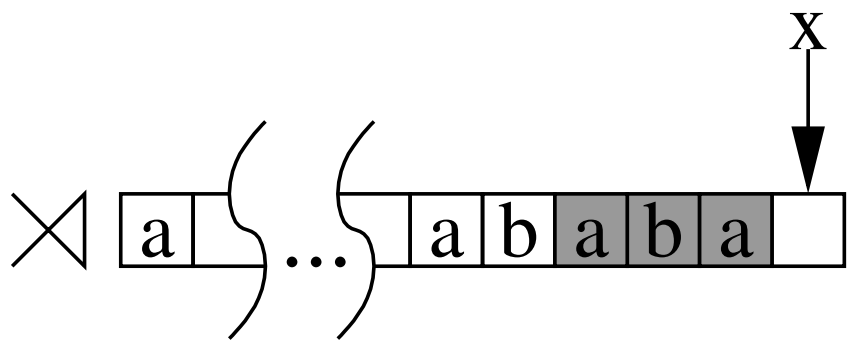


Local String-to-String Functions

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

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

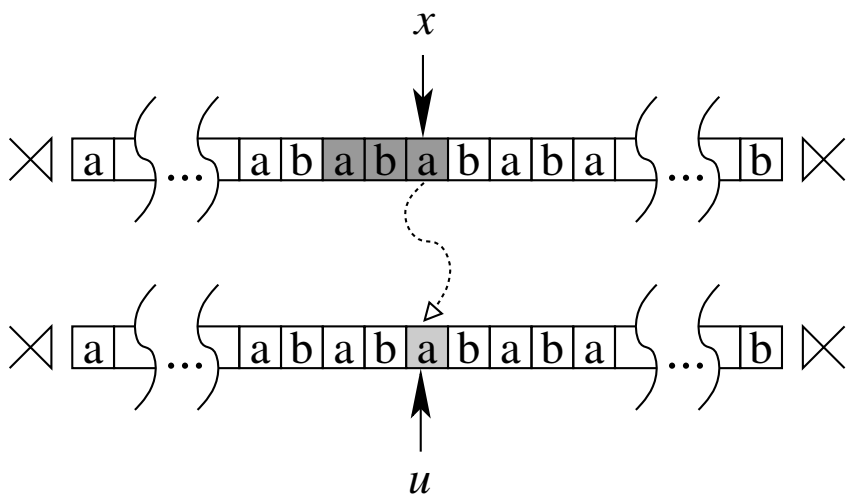
- The probability of the next item only depends on the previous k symbols.



Strictly Local Functions

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

Input Strictly Local

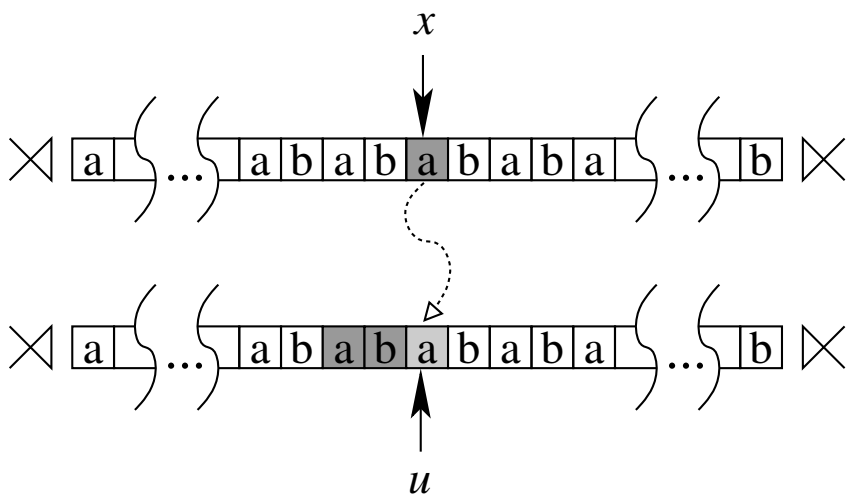


(Chandlee 2014, Chandlee et al. 2014, 2015)

Strictly Local Functions

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

Output Strictly Local

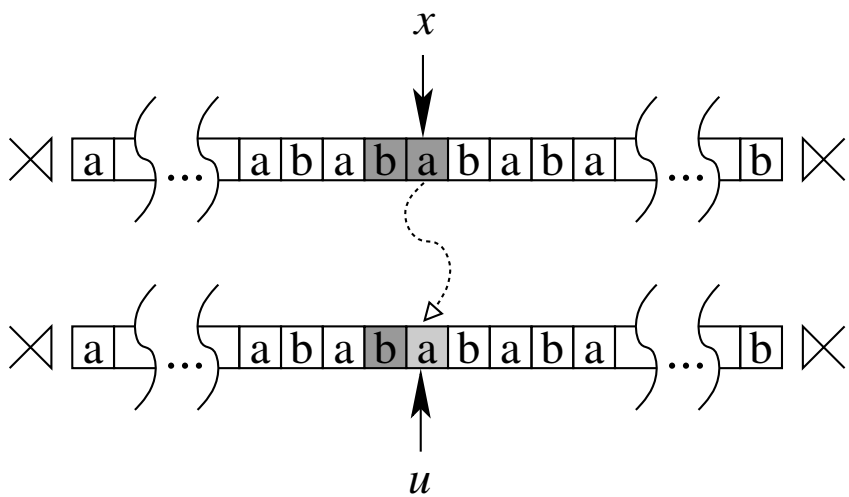


(Chandlee 2014, Chandlee et al. 2014, 2015)

Strictly Local Functions

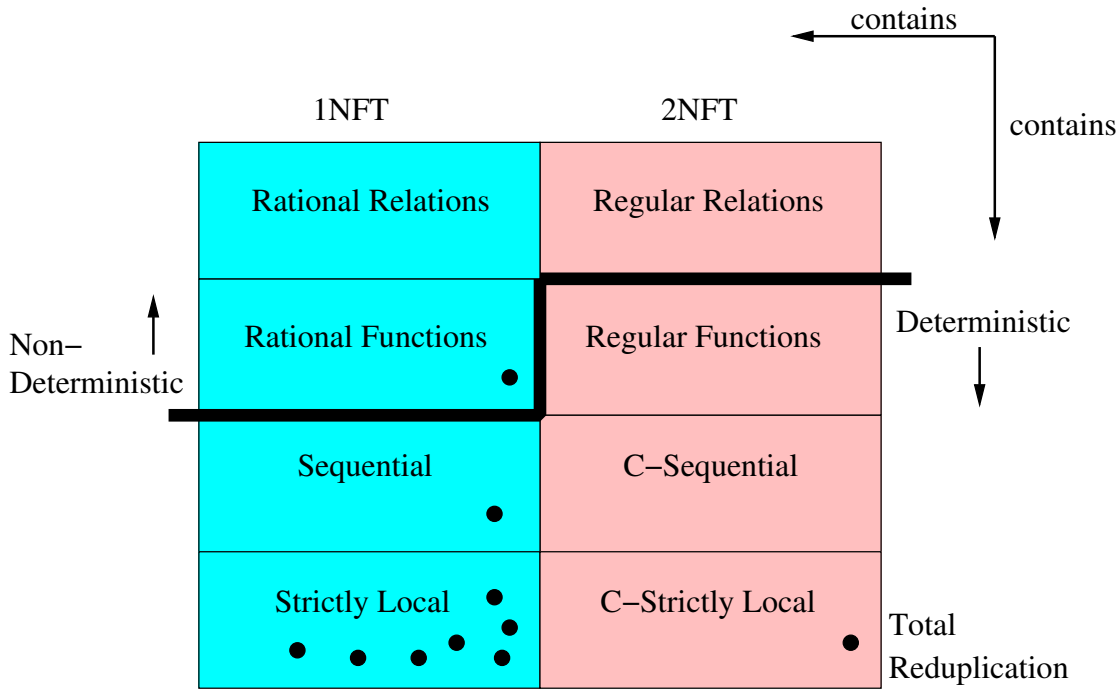
Chandlee develops the same idea in the context of string rewriting.

Input-Output Strictly Local



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

Strictly Local Functions



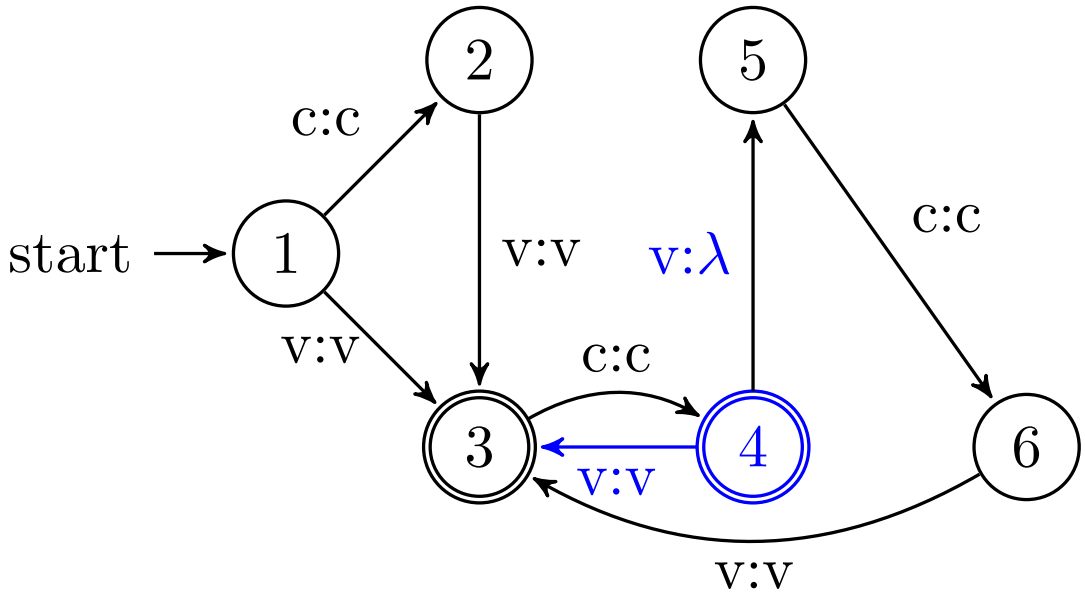
I group all of these types here as "strictly local functions."

- They cover quite a bit of the typology of morphology and phonology (Chandlee 2017, Chandlee and Heinz 2018)
- They are learnable more or less the same way n-gram models are (Chandlee et al. 2014, 2015)
- *Obligatory* iterative French schwa-deletion is Output-Strictly Local!

Part IV

But deterministic on the Input, not the Output!

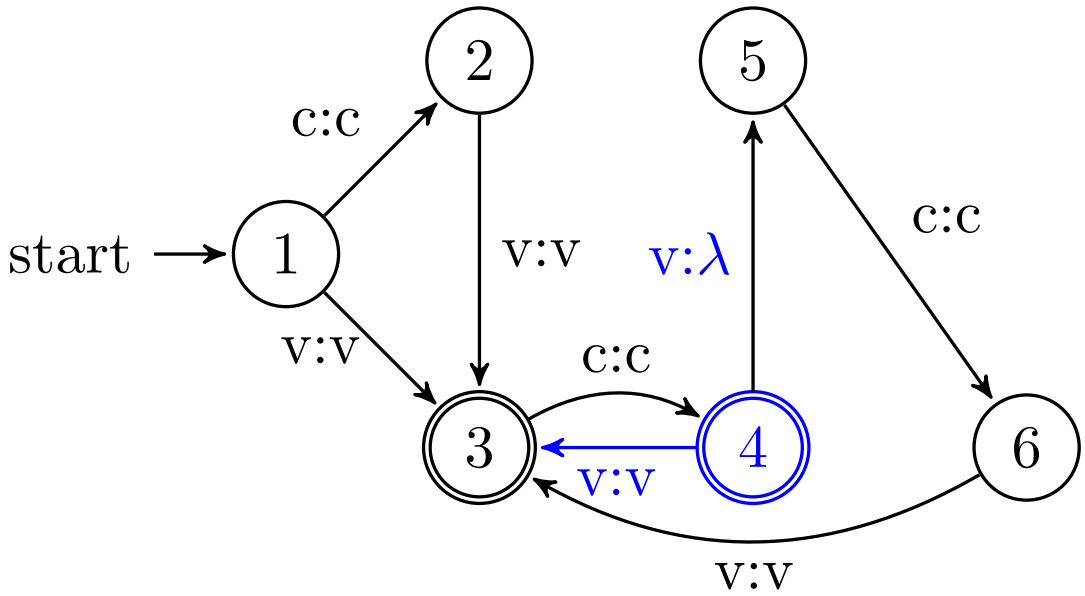
The Challenge: Output-oriented Optionality



- The output determines the state!
- $4 \xrightarrow{\text{v:}\{\text{v},\lambda\}} ??$
- For deterministic transducers, the next state is necessarily determined by the input symbol!

The Challenge: Output-oriented Optionality

Let's call this transducer **T**.



- The output determines the state!
- $4 \xrightarrow{\text{v:}\{\text{v},\lambda\}} ??$
- For deterministic transducers, the next state is necessarily determined by the input symbol!

Resolving The Challenge: Examine examples of the transformation

/cvcv/	/cvcvcv/	/cvcvcvcv/	/cvcvcvcvcv/	
cvcv	cvcvcv	cvcvcvcv	cvcvcvcvcv	faithful
	cvccv	cvccvcv	cvccvcvcv	2nd vowel deletes
		cvcvccv	cvcvccvcv	3rd vowel deletes
			cvccvccv	2nd, 4th vowels delete

Resolving The Challenge: Examine examples of the transformation

/cvcv/	/cvcvcv/	/cvcvcvcv/	/cvcvcvcvcv/	
cvcv	cvcvcv	cvcvcvcv	cvcvcvcvcv	faithful
	cvccv	cvccvcv	cvccvcvcv	2nd vowel deletes
		cvcvccv	cvcvccvcv	3rd vowel deletes
			cvccvccv	2nd, 4th vowels delete

Observations:

- No complex syllable margins!

Resolving The Challenge: What would Kisseberth say?

Kisseberth 1970: 304-305

By making . . . rules meet two conditions (one relating to the form of the input string and the other relating to the form of the output string; one relating to a single rule, the other relating to all the rules in the grammar), we are able to write the vowel deletion rules in the intuitively correct fashion. We do not have to mention in the rules themselves that they cannot yield unpermitted clusters. We state this fact once in the form of a derivational constraint.

Resolving The Challenge: What would OT say?

*Syllable,*Complex $\gg$ Max-V

(Prince and Smolensky 1993, 2004, Zoll 1993, 1996, deLacy 1999, Gouskova 2003)

Resolving The Challenge: Factoring transducer T

$$T = T_1 \circ T_2$$

where

- T_1 is a transducer which optionally deletes vowels.
- T_2 is a phonotactic constraint on complex syllable margins.

Resolving The Challenge: Factoring transducer T

$$T = T_1 \circ T_2$$

where

- T_1 is a transducer which optionally deletes vowels.
- T_2 is a phonotactic constraint on complex syllable margins.

T_1 is Input Strictly Local!

Resolving The Challenge: Factoring transducer T

$$T = T_1 \circ T_2$$

where

- T_1 is a transducer which optionally deletes vowels.
- T_2 is a phonotactic constraint on complex syllable margins.

Both T_1 and T_2 can be learned from examples!

Part V

Learning Deterministic Optional Processes

Resolving The Challenge: Learning T_2

- The constraint *Complex is a Strictly 3-Local constraint and can be learned by any SL3 learner.

  (Garcia et al. 1990, Heinz 2007, et seq., Chandlee et al. 2019, Aksenova 2020)

Resolving The Challenge: Learning T_2

- The constraint *Complex is a Strictly 3-Local constraint and can be learned by any SL3 learner.

  (Garcia et al. 1990, Heinz 2007, et seq., Chandlee et al. 2019, Aksenova 2020)

- On outputs like those shown above, these algorithms return *Complex constraint by forbidding these substrings:

$$\{\text{ccc},\ \rtimes\text{cc},\ \text{cc}\ltimes\}$$

Resolving The Challenge: Learning T_1

The basic idea synthesizes results from Jardine et al. (2014) ...

- The deterministic structure of T_1 is known a priori. It is k-ISL for some k.
- The outputs of the transitions can be determined by mapping/aligning the data along the deterministic paths of the transducer.
- This is accomplished with the principles of "onwardness" and the "longest common prefix."

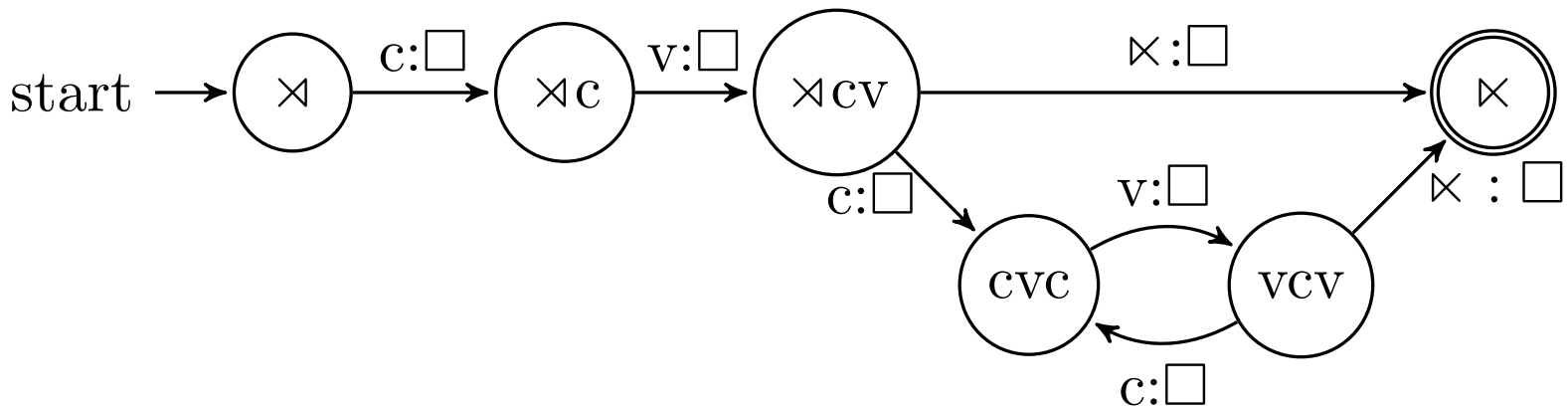
and Beros and de la Higuera (2016).

- The principles of "onwardness" and the "longest common prefix" can be lifted to multiple outputs provided the output sets of strings on the transitions are "pairwise incomparable."

Resolving The Challenge: Learning T_1

Strategy: Learn an Input-based function anyway and filter the outputs with phonotactic constraints (T_2).

Input Strictly Local Transducer with 4-size window



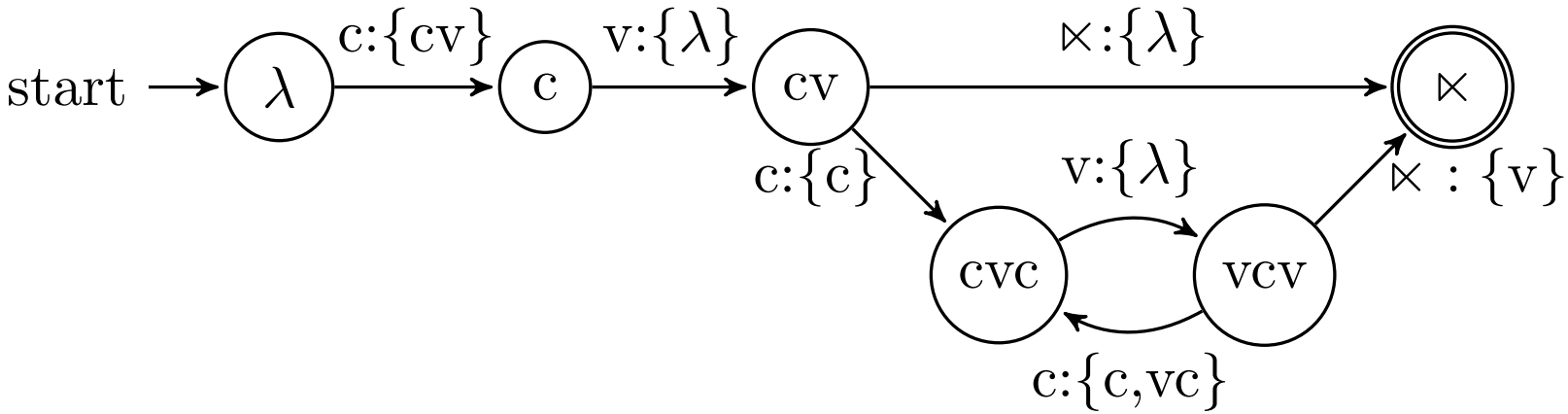
Before Learning

Resolving The Challenge: Learning T_1

A balancing act when mapping the data onto the transitions.

1. Push stringsets forward to output to ensure "onwardness."
2. Push stringsets back to ensure "pairwise incomparability."

Input Strictly Local Transducer with 4-size window



After Learning

Part V

Summary (The End)

Conclusion

1. It is important to understand how morpho-phonological processes fit into the landscape of string-to-string functions.

Conclusion

1. It is important to understand how morpho-phonological processes fit into the landscape of string-to-string functions.
2. Optional processes can be deterministic.
   (Multi-valued $\not\Rightarrow$ non-deterministic.)

Conclusion

1. It is important to understand how morpho-phonological processes fit into the landscape of string-to-string functions.
2. Optional processes can be deterministic.
   (Multi-valued $\not\Rightarrow$ non-deterministic.)
3. Non-decomposed, output-oriented, iterative, optional processes are non-deterministic.

Conclusion

1. It is important to understand how morpho-phonological processes fit into the landscape of string-to-string functions.
2. Optional processes can be deterministic.
   (Multi-valued $\not\Rightarrow$ non-deterministic.)
3. Non-decomposed, output-oriented, iterative, optional processes are non-deterministic.
4. But they can be factored into a deterministic process which overgenerates and a constraint which filters out the unwanted overgenerates.

$$T = T_1 \circ T_2$$

Conclusion

1. It is important to understand how morpho-phonological processes fit into the landscape of string-to-string functions.
2. Optional processes can be deterministic.
   (Multi-valued $\not\to$ non-deterministic.)
3. Non-decomposed, output-oriented, iterative, optional processes are non-deterministic.
4. But they can be factored into a deterministic process which overgenerates and a constraint which filters out the unwanted overgenerates.

$$T = T_1 \circ T_2$$

5. T_2 can be learned with existing grammatical inference methods.

Conclusion

1. It is important to understand how morpho-phonological processes fit into the landscape of string-to-string functions.
2. Optional processes can be deterministic.
   (Multi-valued $\not\to$ non-deterministic.)
3. Non-decomposed, output-oriented, iterative, optional processes are non-deterministic.
4. But they can be factored into a deterministic process which overgenerates and a constraint which filters out the unwanted overgenerates.

$$T = T_1 \circ T_2$$

5. T_2 can be learned with existing grammatical inference methods.
6. T_1 appears to be learnable with a synthesis of recent results in grammatical inference.

Discussion

1. Formal proof of correctness of the algorithm for learning classes of structured multi-valued functions is in progress.

Discussion

1. Formal proof of correctness of the algorithm for learning classes of structured multi-valued functions is in progress.
2. Probabilities can be appended to the outputs for learning classes of functions $\Sigma^* \rightarrow \mathrm{P(FIN)}$.

Discussion

1. Formal proof of correctness of the algorithm for learning classes of structured multi-valued functions is in progress.
2. Probabilities can be appended to the outputs for learning classes of functions $\Sigma^* \rightarrow \mathrm{P(FIN)}$.
3. We hope to apply to other problems:
    1. learning URs and phonological grammars simultaneously
    2. sociolinguistic variation
    3. NLP problems such as G2P, P2G, and so on.

Thank

You

Appendix: Pairwise Incomparability, Onwardness, Longest Common Prefix

Pairwise Incomparability

Informally, a finite set of strings S is pairwise incomparable provided, for each pair of distinct strings drawn from S, neither is a proper prefix of the other.

Pairwise Incomparability

Informally, a finite set of strings S is pairwise incomparable provided, for each pair of distinct strings drawn from S, neither is a proper prefix of the other.

Formally

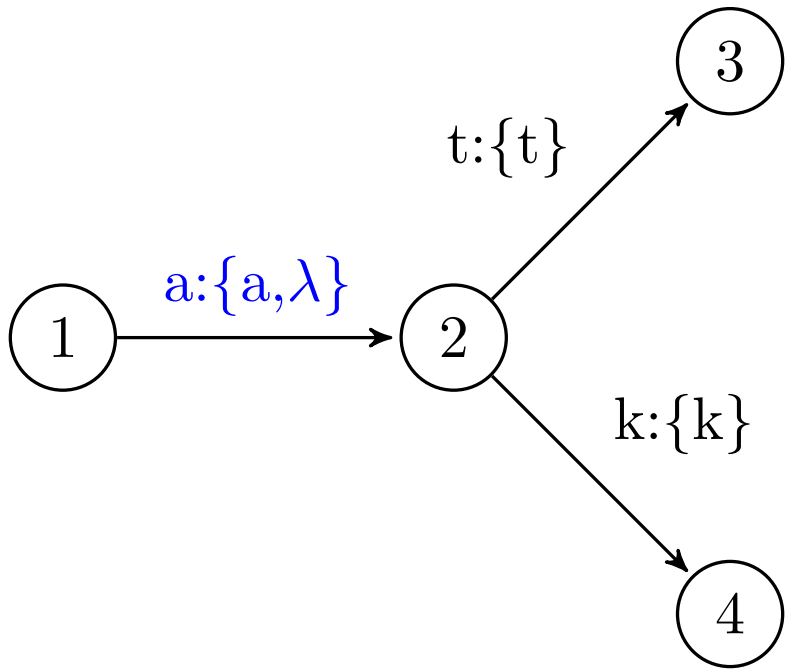
- We write $x < y$ if there is a string $z \neq \lambda$ such that $y = xz$.
- If $x = y$, $x < y$ or $x > y$ then say x and y are **comparable**.
- Otherwise, we say that x and y are **incomparable** and write $x \perp y$.
- A finite set of strings S is **pairwise incomparable** iff for each $x, y \in S$, we have $x \perp y$.

Pairwise Incomparability

- Beros and de la Higuera (2016) require the output sets on each transition to be pairwise incomparable.
- This allows them to establish a minimal, canonical form for a class of functions $\Sigma^* \rightarrow \text{FIN}$.
- How they do this is informative, and I will return to it momentarily.

Enforcing Pairwise-incomparability

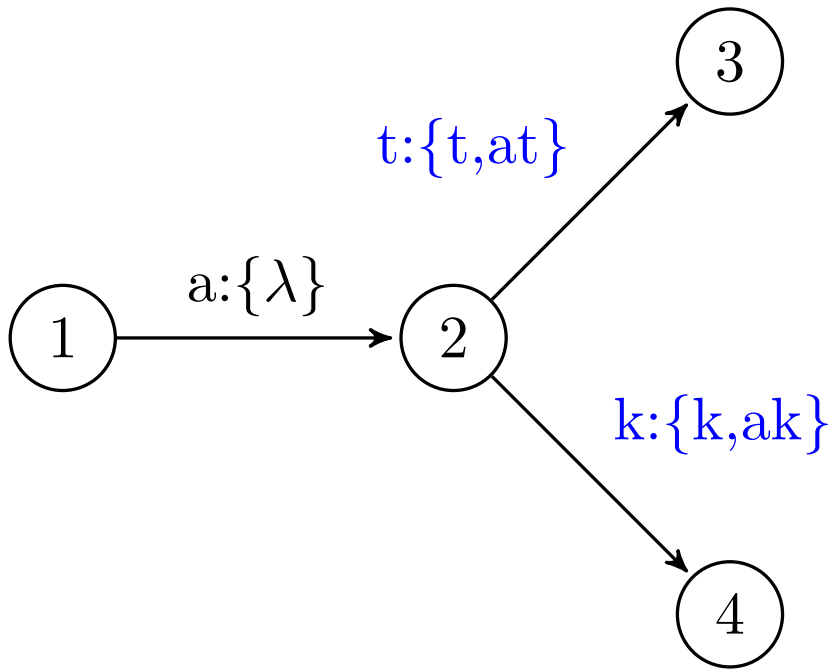
Optional /a/-deletion (Deterministic)



- $S = \{a,\lambda\}$ is not pairwise incomparable!

Ensuring Pairwise Incomparability

Optional /a/-deletion
(Deterministic & Pairwise Incomparable)



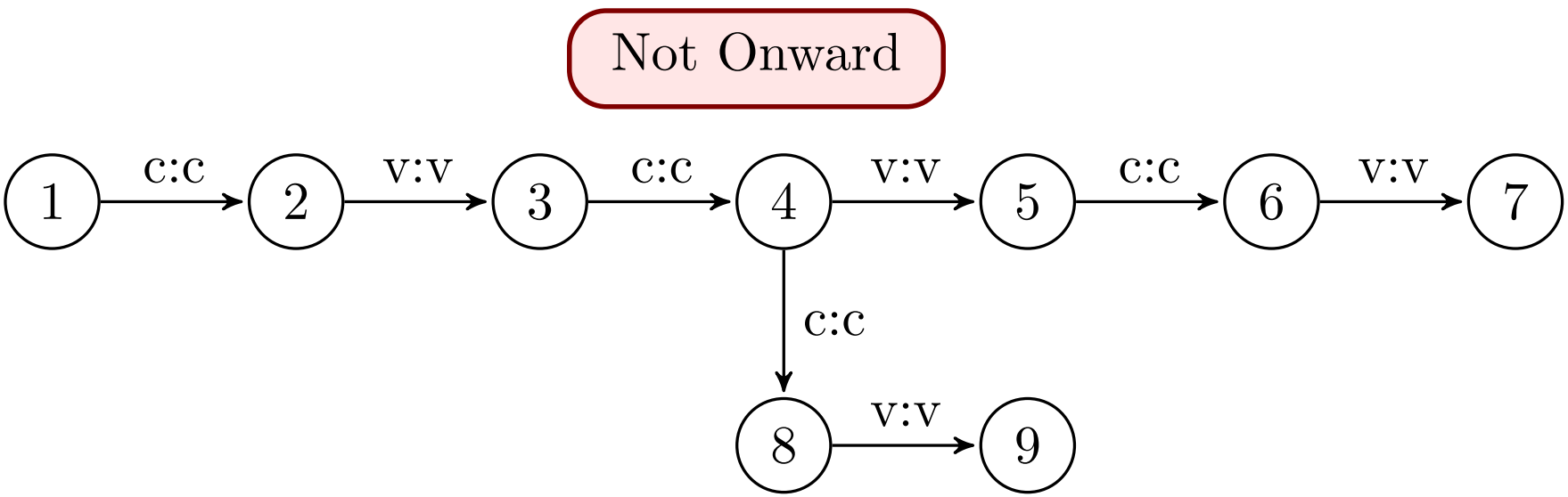
- By 'pushing' outputs, we can get pairwise incomparability!

Onwardness

- Recall Beros and de la Higuera used pairwise-incomparability to reveal canonical forms for deterministic functions with finite stringsets on the output transitions.
- One way to define a canonical form for deterministic transducers is to require outputs be produced as early as possible. This has been called 'onwardness' (Oncina et al. 1993).

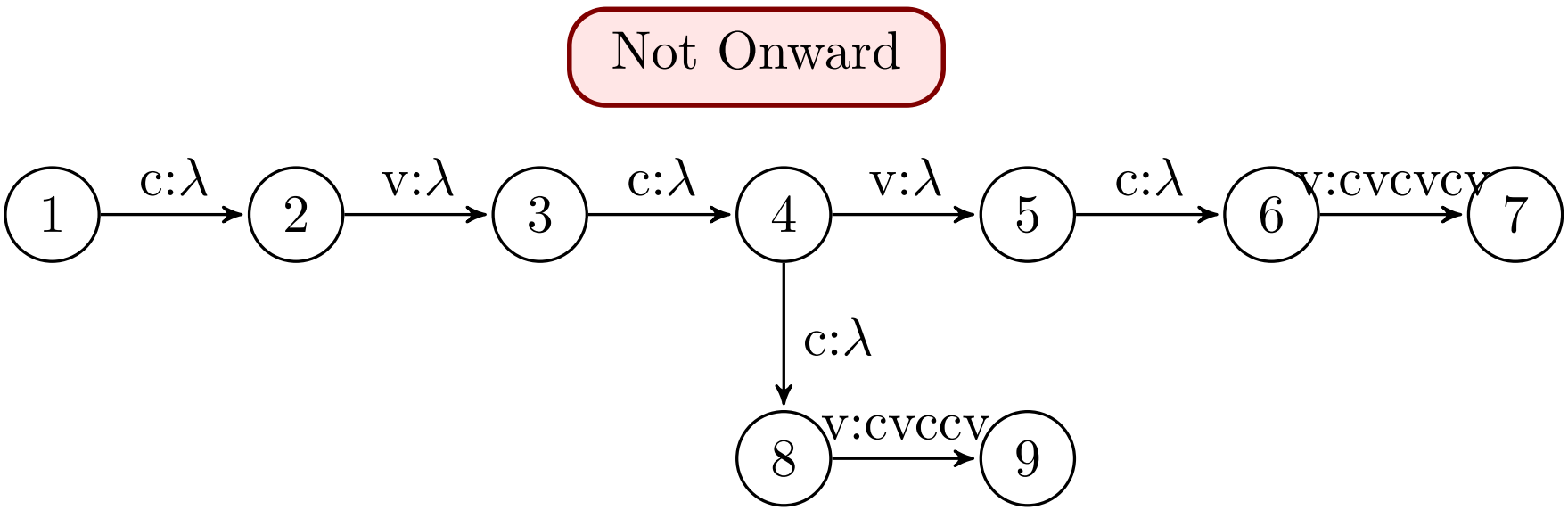
Onwardness

- Recall Beros and de la Higuera used pairwise-incomparability to reveal canonical forms for deterministic functions with finite stringsets on the output transitions.
- One way to define a canonical form for deterministic transducers is to require outputs be produced as early as possible. This has been called 'onwardness' (Oncina et al. 1993).



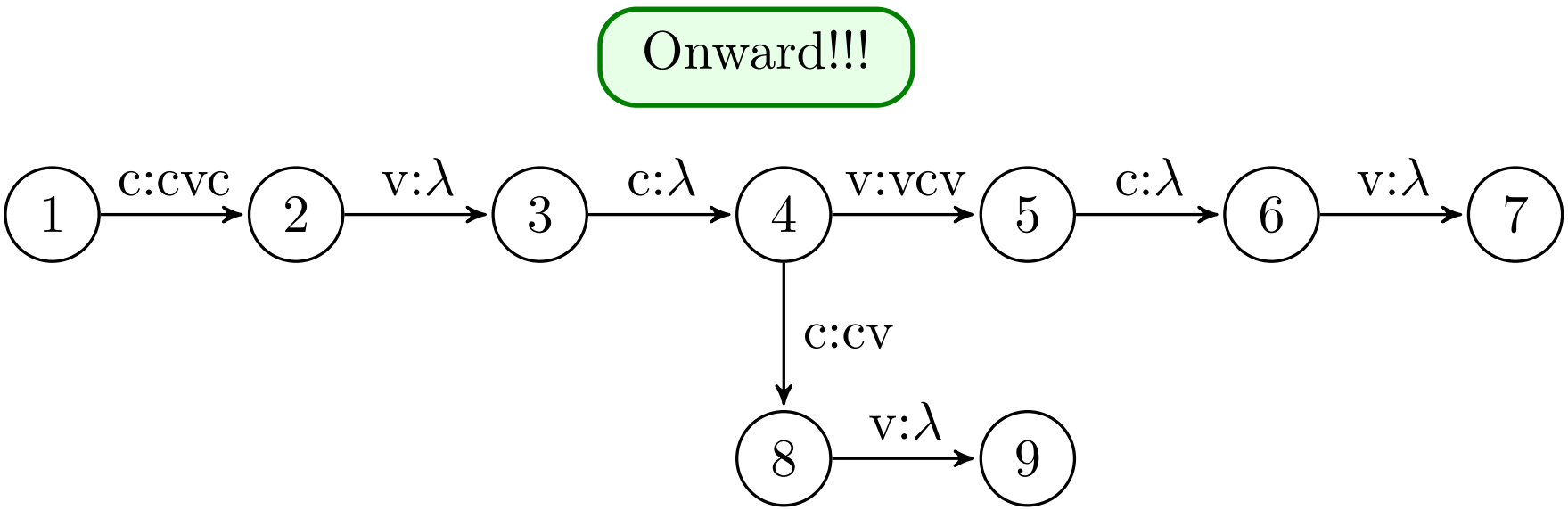
Onwardness

- Recall Beros and de la Higuera used pairwise-incomparability to reveal canonical forms for deterministic functions with finite stringsets on the output transitions.
- One way to define a canonical form for deterministic transducers is to require outputs be produced as early as possible. This has been called 'onwardness' (Oncina et al. 1993).



Onwardness

- Recall Beros and de la Higuera used pairwise-incomparability to reveal canonical forms for deterministic functions with finite stringsets on the output transitions.
- One way to define a canonical form for deterministic transducers is to require outputs be produced as early as possible. This has been called 'onwardness' (Oncina et al. 1993).



Onwardness for String Transducers

- The **longest common prefix** is used to make string transducers onward.

$$\texttt{lcp}\left(\left\{\begin{array}{l} c\ v\ c\ v\ c\ v \\ c\ v\ c\ c\ v \end{array}\right\}\right) = cvc$$

Onwardness for String Transducers

- The **longest common prefix** is used to make string transducers onward.

$$\texttt{lcp}\left(\left\{\begin{array}{l} c\ v\ c\ v\ c\ v \\ c\ v\ c\ c\ v \end{array}\right\}\right) = cvc$$

- We strip off the `lcp` of the other strings to get the remainder.

$$(cvc)^{-1}\left\{\begin{array}{l} c\ v\ c\ v\ c\ v \\ c\ v\ c\ c\ v \end{array}\right\} = \left\{\begin{array}{l} v\ c\ v \\ c\ v \end{array}\right\}$$

Onwardness for String Transducers

- The **longest common prefix** is used to make string transducers onward.

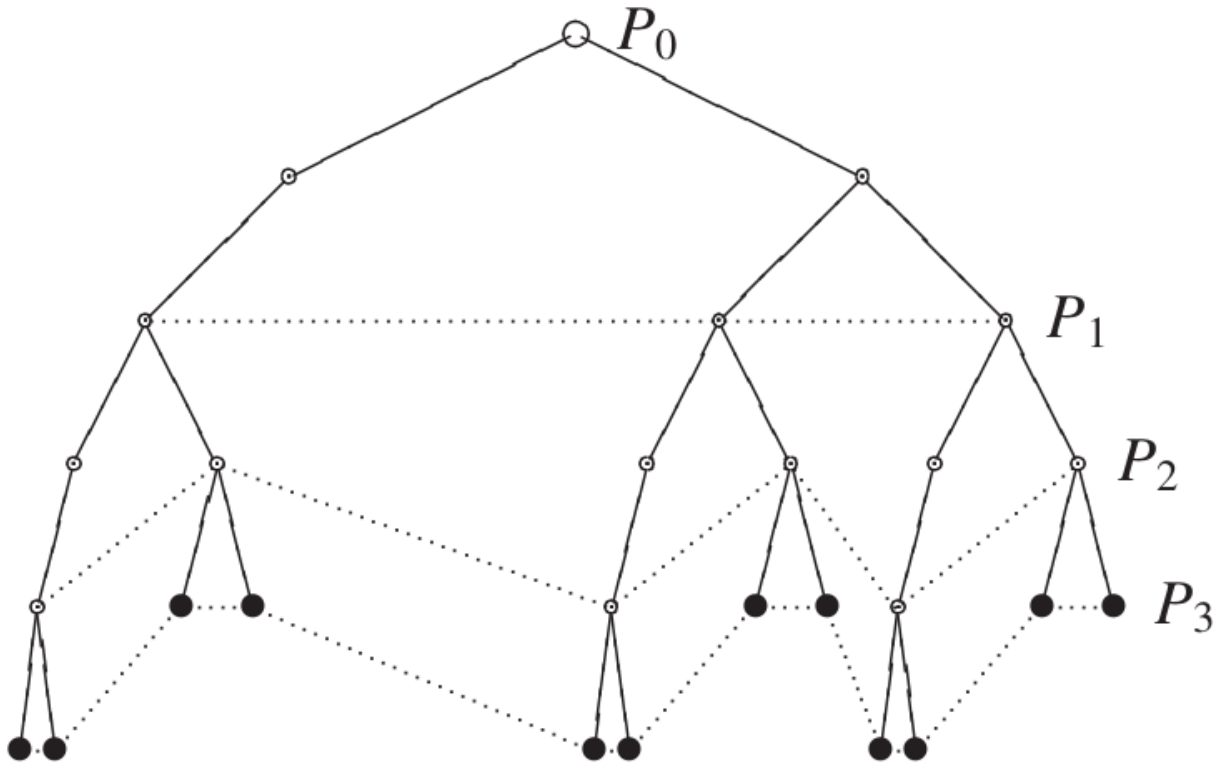
$$\texttt{lcp}\left(\left\{\begin{array}{l} c\ v\ c\ v\ c\ v \\ c\ v\ c\ c\ v \end{array}\right\}\right) = cvc$$

- The same idea is used in Jardine et al. for learning:

$$\text{For } q \xrightarrow{a:\square} q' : \square = \texttt{lcp}(w_q\Sigma^*)^{-1}\texttt{lcp}(wa\Sigma^*)$$

Onwardness for Finite Stringset Transducers

- The **maximal-length, pairwise-incomparable shared prefixes** are used to make finite stringset transducers onward.

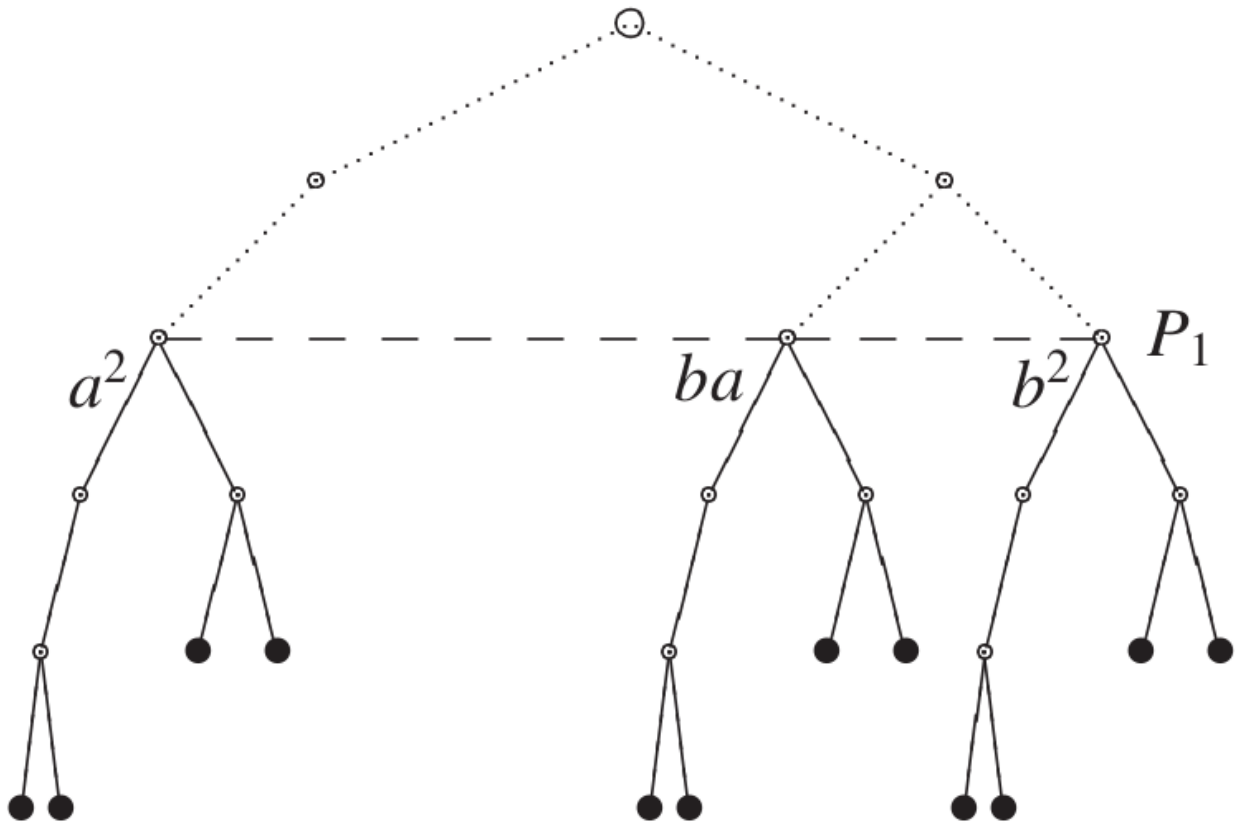


- For $q \xrightarrow{a:\square} q', \square = \texttt{mlpisp}(w_q\Sigma^*)^{-1}\texttt{mlpisp}(wa\Sigma^*)$

(pics: Beros and de la Higuera 2016)

Onwardness for Finite Stringset Transducers

- The **maximal-length, pairwise-incomparable shared prefixes** are used to make finite stringset transducers onward.



- For $q \xrightarrow{a:\square} q', \square = \texttt{mlpisp}(w_q\Sigma^*)^{-1}\texttt{mlpisp}(wa\Sigma^*)$

(pics: Beros and de la Higuera 2016)