Deterministic Analyses of Optional Processes

Jeffrey Heinz

Rutgers University
November 22, 2019

Part I

What am I talking about?

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)

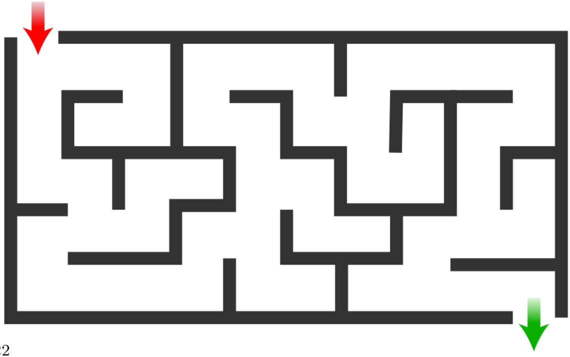
Deterministic transformations in phonology

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. 2019)

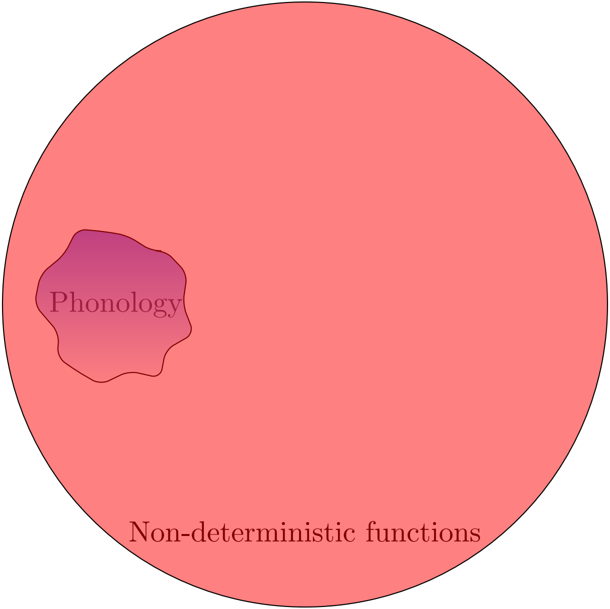
What is 'determinism'? Why does it matter?

- 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.

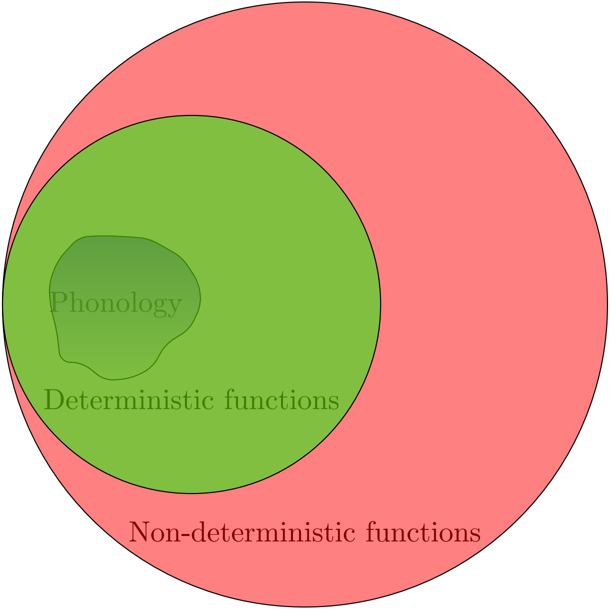


Why does it matter?

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. 2019:19

...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).
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 and Eric Schieferstein

Part II

Optionality and Determinism

Iterative Optionality

Vaux 2008, p. 43

(14) French schwa deletion

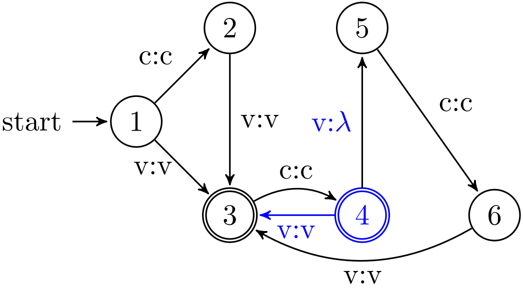
a. ə→ Ø / V (#) C _, L→R , optional across #

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

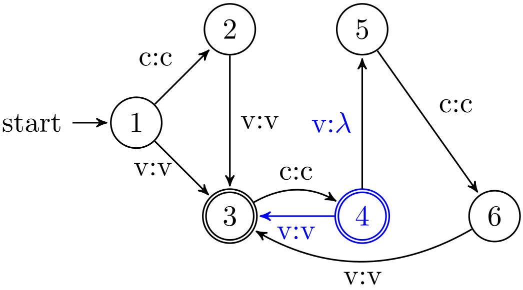
ãvidtəldəmãde
ãvidtələdəmãde
ãvidtələdmãde
ãvidətələdmãde
ãvidətlədmãde
ãvidətlədəmãde
ãvidətəldəmãde
ãvidətələdəmãde

Optional Syncope as a finite-state function

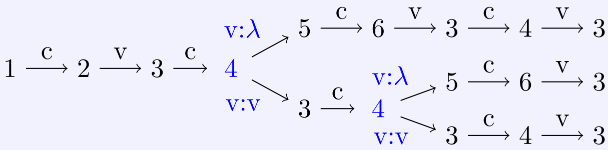
V → ∅ / VC __ CV (applying left-to-right)



Optional Syncope as a 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?

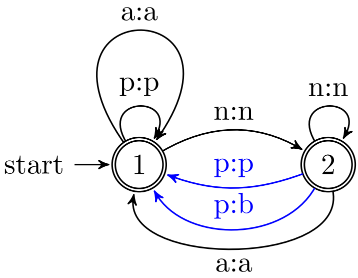
- A function is **single-valued** if there is at most one output for each input.
- What is the relationship between single-valuedness and determinism?
    1. Does single-valuedness imply determinism?
    2. Does determinism imply single-valuedness?

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…

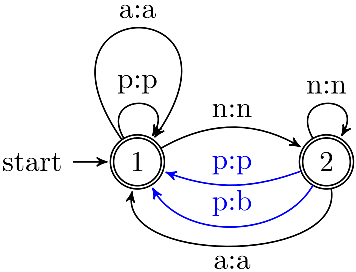
Deterministic FSTs with Language Monoids

Optional Post-nasal Voicing (Non-deterministic)

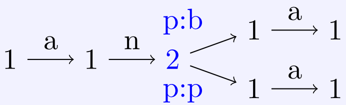


Deterministic FSTs with Language Monoids

Optional Post-nasal Voicing
(Non-deterministic)

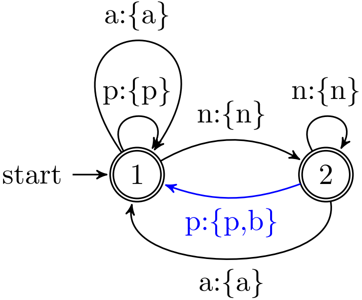


/ a n p a /



Deterministic FSTs with Language Monoids

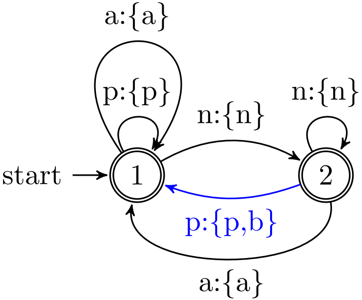
Optional Post-nasal Voicing (Deterministic)



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

Deterministic FSTs with Language Monoids

Optional Post-nasal Voicing (Deterministic)

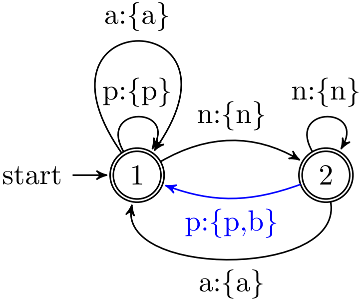


/ a n p a /

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

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

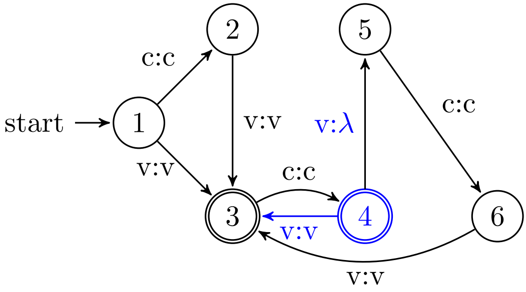
Name	K	$\otimes$	1	
String	Σ^*	$\cdot$	λ	$\Sigma^* \to \Sigma^*$
Boolean	$\{T, F\}$	$\wedge$	`true`	$\Sigma^* \to \{T, F\}$
Natural	$\mathbb{N}$	$+$	0	$\Sigma^* \to \mathbb{N}$
Real Interval	$[0, 1]$	$\times$	1	$\Sigma^* \to [0, 1]$
FIN	$\{L \subseteq \Sigma^* \mid L \text{ finite}\}$	$\cdot$	$\{\lambda\}$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

- 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).

Part III

But it's not that simple. . .

Issue #1: 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!

Issue #2: 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.

Issue #2: 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

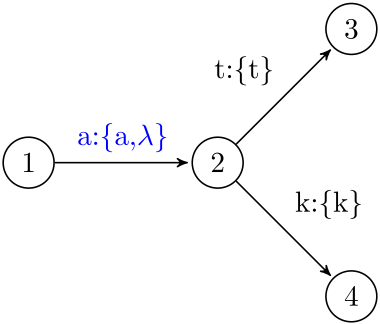
- We write $x < y$ if there is a string $z \neq \lambda$ such that $y = xz$.
- If $x = y$, $x < y$ or $x > y$ then say x and y are **comparable**.
- Otherwise, we say that x and y are **incomparable** and write $x \perp y$.
- A finite set of strings S is **pairwise incomparable** iff for each $x, y \in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

Issue #2: 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 $\Sigma^* \to \mathrm{FIN}$.
- How they do this is informative, and I will return to it momentarily.

Enforcing Pairwise-incomparability

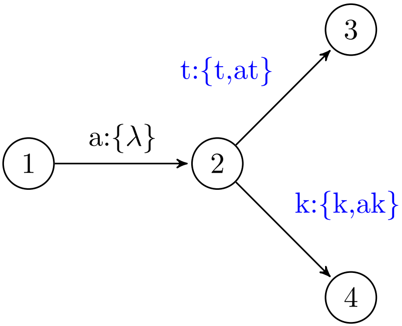
Optional /a/-deletion
(Deterministic)



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

Resolving Issue #2: Pairwise Incomparability

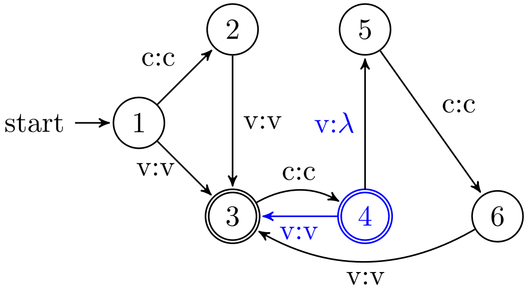
Optional /a/-deletion
(Deterministic & Pairwise Incomparable)



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

Issue #1: 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 Issue #2: 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 Issue #2: 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 Issue #2: 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 Issue #2: What would OT say?

*Syllable,*Complex ≫ Max-V

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

Resolving Issue #2: 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 Issue #2: 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 IV

Learning Deterministic Optional Processes

Resolving Issue #2: 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)

Resolving Issue #2: 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)

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

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

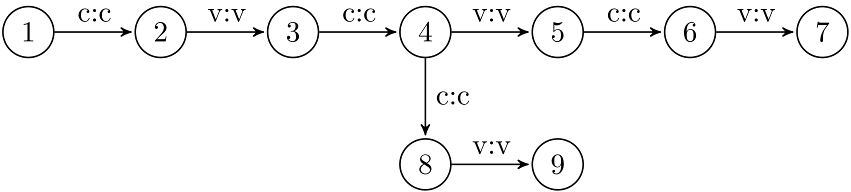
Onwardness

- 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).

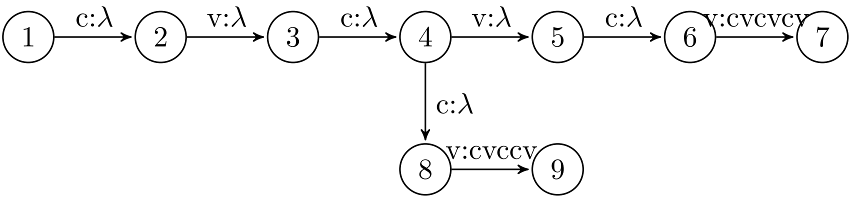
Not Onward



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).

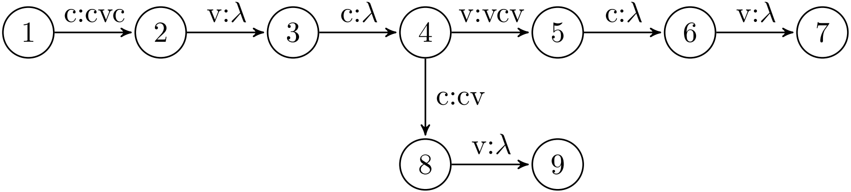
Not Onward



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).

Onward!!!



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.

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

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

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

Onwardness for String Transducers

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

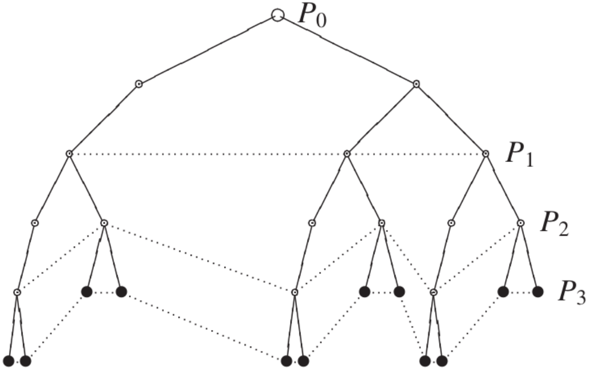
$$\texttt{lcp}\left(\left\{\begin{array}{l} \color{blue}{c\ v\ c}\ v\ c\ v \\ \color{blue}{c\ v\ c}\ c\ v \end{array}\right\}\right) = cvc$$

- 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.



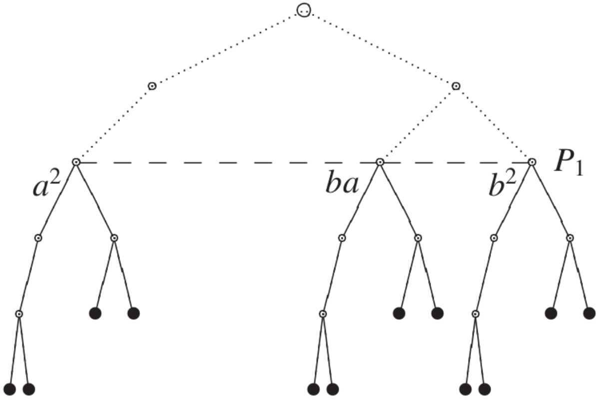
-

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

(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.



-

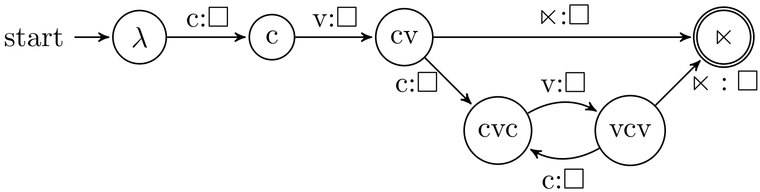
$$\text{For } q \xrightarrow{a:\square} q', \square = \texttt{mlpisp}(w_q\Sigma^*)^{-1}\texttt{mlpisp}(wa\Sigma^*)$$

(pics: Beros and de la Higuera 2016)

Resolving Issue #2: 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

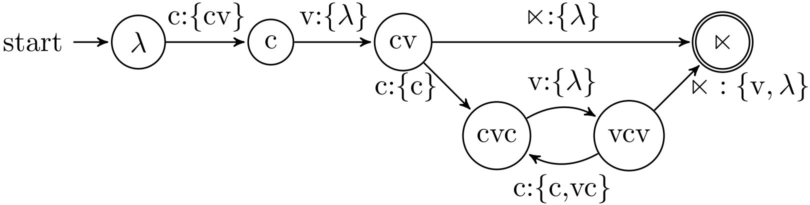


Before Learning

Resolving Issue #2: Learning T_1

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. Optional processes can be deterministic.
   (Multi-valued $\not\rightarrow$ non-deterministic.)

Conclusion

1. Optional processes can be deterministic.
   (Multi-valued $\not\rightarrow$ non-deterministic.)
2. Non-decomposed, output-oriented, optional processes are non-deterministic.

Conclusion

1. Optional processes can be deterministic.
   (Multi-valued $\not\rightarrow$ non-deterministic.)
2. Non-decomposed, output-oriented, optional processes are non-deterministic.
3. 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. Optional processes can be deterministic.
   (Multi-valued $\not\rightarrow$ non-deterministic.)
2. Non-decomposed, output-oriented, optional processes are non-deterministic.
3. But they can be factored into a deterministic process which overgenerates and a constraint which filters out the unwanted overgenerates.

$$T = T_1 \circ T_2$$

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

Conclusion

1. Optional processes can be deterministic. (Multi-valued $\not\rightarrow$ non-deterministic.)
2. Non-decomposed, output-oriented, optional processes are non-deterministic.
3. But they can be factored into a deterministic process which overgenerates and a constraint which filters out the unwanted overgenerates.

$$T = T_1 \circ T_2$$

4. T_2 can be learned with existing grammatical inference methods.
5. 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 $\Sigma^* \to P(\mathrm{FIN})$.
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