Nonderived Environment Blocking: A Computational Account

HAVERFORD
COLLEGE

Jane Chandlee

LSA 2020 | New Orleans

Overview

- Present a computational analysis of nonderived environment blocking (NDEB)
  - Classic example from Finnish, with both types of derived environment (morphological and phonological)
- NDEB can be modeled with an input strictly local (ISL) function
  - Most restrictive in terms of computational complexity
- Analysis makes use of a logical characterization of ISL, using Boolean Monadic Recursive Schemes

Finnish (Kiparsky, 1973, 1993)

	/halut-i/	/vete/	/tila/
e → i / __ #	—	veti	—
t → s / __ i	halusi	vesi	BLOCK
	[halusi]	[vesi]	[tila]
	'want-PST'	'water-NOMSG'	'room-NOMSG'

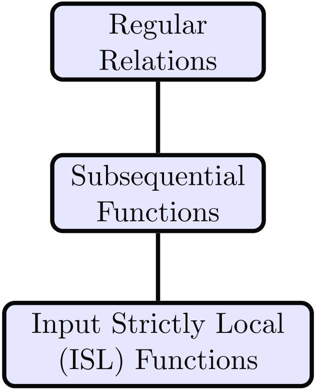
Polish (Rubach, 1984; Lubowicz, 2002)

	/krok-i-ć/	/śńeg-ic-a/	/banʤ-o/
Palatalization	kroʧ-i-ć	śńeʤ-ic-a	—
Spirantization	—	śńeʒ-ic-a	BLOCK
	[kroʧɨć]	[śńeʒɨca]	[banʤo]
	'to step'	'snowstorm'	'banjo'

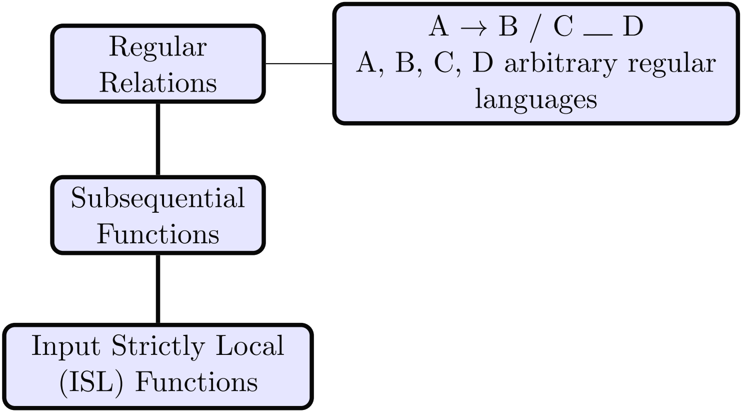
Previous analyses

- Strict Cycle Condition (Mascaró, 1976; Halle, 1978; Kiparsky, 1982)
- Underspecification (Kiparsky, 1993)
- Neighborhood preservation (Itô and Mester, 1996)
- Output-output faithfulness (Burzio, 2000)
- Structural immunity (Inkelas, 2001)
- Constraint conjunction (Lubowicz, 2002)
- Comparative markedness constraints (McCarthy, 2003)
- Precedence constraints (Wolf, 2008)

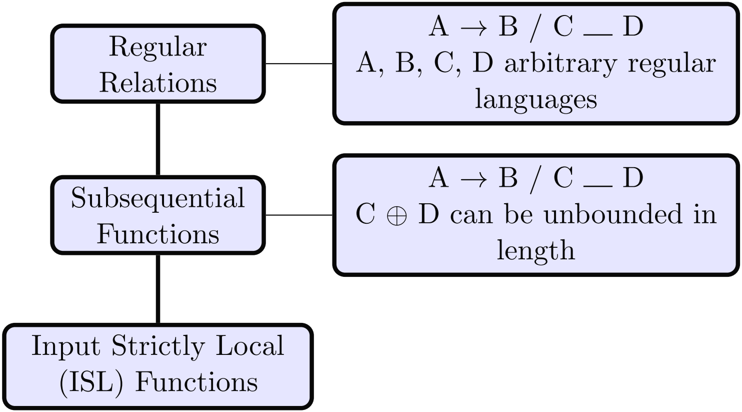
- Processes are *maps*: /halut-i/ $\mapsto$ [halusi]
- The *computational complexity* of a map relates to the *amount of information* needed to determine the output string for a given input string.
- Types of processes vary in computational complexity.


Regular Relations
Subsequential Functions
Input Strictly Local (ISL) Functions

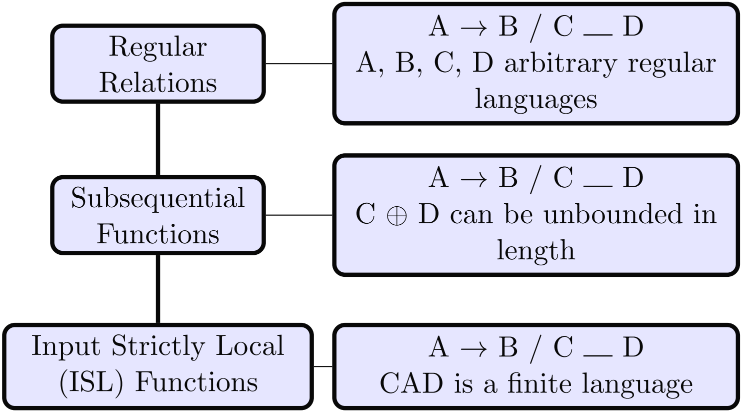
Hierarchy of function classes



Hierarchy of function classes



Hierarchy of function classes



Input strictly local (ISL) functions

- Available/needed information is an input substring of bounded length.
- E.g., Finnish word-final vowel raising is 2-ISL:

j	o	k	e	#
j	o	k	λ	i

- ISL functions can model multiple processes at once.
- Do process *interactions* increase complexity?
- For a range of opaque interactions, the answer is no (Chandlee et al., 2018)

(1) Counterbleeding in Yowlumne (McCarthy, 1999)

a. Long vowels lower

b. Long vowels in closed syllables shorten

c. /miːk-hin/ ↦ [mekhin], ‘swallowed’

m	iː	k	h	i	n
m	λ	λ	ekh	i	n

- More generally, an interaction is ISL if differences in output strings are *fully determined* by input substrings.
- NDEB as an interaction: feeding + blocking
  - Also ISL!
- Logical characterization: Boolean monadic recursive scheme (BMRS) (Bhaskar et al. (to appear))
  - Based on recursive program schemes of Moschovakis (2019)
  - Restriction to ISL: non-recursive

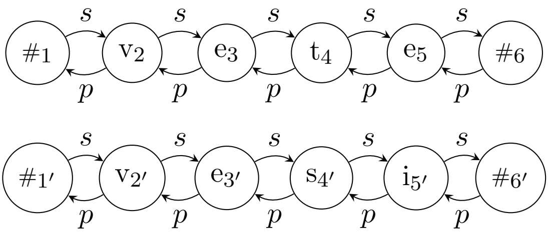
	/halut-i/	/vete/	/tila/
e → i / __ #	—	veti	—
t → s / __ i	halusi	vesi	BLOCK
	[halusi]	[vesi]	[tila]

	/halut-i/	/vete#/	/tila/
	[halusi]	[vesi]	[tila]

- Logical characterization of a function

Input:

Output:



Graph transduction

Input:	#	v	e	t	e	#
	1	2	3	4	5	6
Output:	#	v	e	s	i	#
	$1'$	$2'$	$3'$	$4'$	$5'$	$6'$

Output predicates

Input:	#	v	e	t	e	#
	1	2	3	4	5	6
Output:	?	?	?	?	?	?
	$1'$	$2'$	$3'$	$4'$	$5'$	$6'$

$\text{i}_o(x) = \ldots$ (under what conditions is an output segment a [i]?)

Input:	#	v	e	t	e	#
	1	2	3	4	5	6
Output:	?	?	?	?	?	?
	1′	2′	3′	4′	5′	6′

$\mathrm{i}_o(x) =$ `if` … `then` … `else` …

Input:	#	v	e	t	e	#
	1	2	3	4	5	6

$$e(x)$$

Input:	#	v	e	t	e	#
	1	2	3	4	5	6

$$\underline{e}\#(x) = \texttt{if}\ \ e(x)\ \ \texttt{then}\ \ \#(s(x))\ \ \texttt{else}\ \ \textsc{False}$$

Output predicates

Input:	#	v	e	t	e	#
	1	2	3	4	5	6
Output:					i	
	1′	2′	3′	4′	5′	6′

$\text{i}_o(x) = \texttt{if}\ \ \underline{\text{e}}\#(\text{x})\ \ \texttt{then}\ \ \textsc{True}\ \ \texttt{else}\ \ \text{i}(x)$

Output predicates

Input:	#	v	e	t	e	#
	1	2	3	4	5	6
Output:				s	i	
	1′	2′	3′	4′	5′	6′

$$\mathrm{s}_o(x) =$$

$\texttt{if}\ \underline{t}{-}i(x)\ \texttt{then}\ \textsc{True}\ \texttt{else}\ \texttt{if}\ \underline{t}e\#(x)\ \texttt{then}\ \textsc{True}\ \texttt{else}\ s(x)$

Input:	#	v	e	t	e	#
	1	2	3	4	5	6
Output:		v	e	s	i	
	$1'$	$2'$	$3'$	$4'$	$5'$	$6'$

Restriction to ISL

- All predicates refer only to bounded contiguous portions of the *input*.
  - No recursion.

- Multiple 'triggers' for assibilation: before /i/ over a morpheme boundary and before word-final /e/.
- Captures role of vowel raising in the assibilation, but without the need for intermediate representations.

Conclusion

- Main contribution: NDEB is not more computationally complex than other locally-conditioned phonological phenomena (i.e., single processes and opaque interactions)
- Morphologically and phonologically derived environments are the same computationally provided we admit the morpheme boundary into the alphabet
- The formal structure of a BMRS analysis limits the kinds of maps it can model.
- Future work: further restrictions likely exist in terms of dependencies on how these structures are filled in.

References I

Bhaskar, S., Chandlee, J., Jardine, A., and Oakden, C. (to appear). Boolean monadic recursive schemes as a logical characterization of the subsequential functions. In *Proceedings of LATA 2020*.

Burzio, L. (2000). Cycles, non-derived-environment blocking, and correspondence. In Dekkers, J., van der Leeuw, F., and van de Weijer, J., editors, *Optimality Theory: Phonology, syntax, and acquisition*, pages 47–87. Oxford: Oxford University Press.

Chandlee, J., Heinz, J., and Jardine, A. (2018). Input strictly local opaque maps. *Phonology*, 35(2):171–205.

Halle, M. (1978). *Formal versus functional considerations in phonology.* Bloomington: Indiana University Linguistics Club.

Inkelas, S. (2001). Phonotactic blocking through structural immunity. In Stiebels, B. and Wunderlich, D., editors, *Lexicon in Focus. Studia grammatica 45.* Berlin: Akademie Verlag.

Itô, J. and Mester, A. (1996). Structural economy and OCP interactions in local domains. In *Proceedings of the Western Conference on Linguistics (WECOL).* University of California, Santa Cruz.

Kiparsky, P. (1973). *Abstractness, opacity, and global rules.* Bloomington: Indiana University Linguistics Club.

References II

Kiparsky, P. (1982). Lexical morphology and phonology. In Yang, I.-S., editor, *Linguistics in the morning calm*, pages 3–91. Seoul: Hanshin.

Kiparsky, P. (1993). Blocking in nonderived environments. In Kaisse, E. M. and Hargus, S., editors, *Phonetics and Phonology 4: Studies in Lexical Phonology*, pages 277–313. San Diego: Academic Press.

Lubowicz, A. (2002). Derived environment effects in Optimality Theory. *Lingua*, 112.

Mascaró, J. (1976). *Catalan phonology and the phonological cycle*. PhD thesis, MIT.

McCarthy, J. J. (1999). Sympathy and phonological opacity. *Phonology*, 16:331–399.

McCarthy, J. J. (2003). Comparative markedness. *Theoretical Linguistics*, 29:1–51.

Moschovakis, Y. (2019). *Abstract recursion and intrinsic complexity, Lecture Notes in Logic*, volume 48. Cambridge University Press.

Rubach, J. (1984). *Cyclic and lexical phonology: the structure of Polish*. Dordrecht:Foris.

Wolf, M. (2008). *Optimal interleaving: Serial phonology–morphology interaction in a constraint-based model*. PhD thesis, UMass.

$$-\underline{i}(x) = \texttt{if}\ \ i(x)\ \ \texttt{then}\ \ -(p(x))\ \ \texttt{else}\ \ \textsc{False}$$

$$\underline{t}-i(x) = \texttt{if}\ \ t(x)\ \ \texttt{then}\ \ -i(s(s(x)))\ \ \texttt{else}\ \ \textsc{False}$$

$$\underline{t}e\#(x) = \texttt{if}\ \ t(x)\ \ \texttt{then}\ \ e\#(s(x))\ \ \texttt{else}\ \ \textsc{False}$$

$\mathrm{t}_o(x) =$
$\texttt{if}\ \underline{\mathrm{t}}\text{-i}(\mathrm{x})\ \texttt{then}\ \textsc{False}\ \texttt{else}\ \texttt{if}\ \underline{t}e\#(x)\ \texttt{then}\ \textsc{False}\ \texttt{else}\ \mathrm{t}(\mathrm{x})$

$\mathrm{e}_o(x) = \texttt{if}\ \underline{\mathrm{e}}\#(\mathrm{x})\ \texttt{then}\ \textsc{False}\ \texttt{else}\ \mathrm{e}(\mathrm{x})$

$\mathrm{v}_o(x) = \mathrm{v}(x)$

$\mathrm{l}_o(x) = \mathrm{l}(x)$

$\mathrm{a}_o(x) = \mathrm{a}(x)$

$\#_o(x) = \textsc{False}$

#	t	i	l	a	t	-	i
	λ	ti	l	a	λ	λ	si

j	o	k	e	#
j	o	k	λ	i

v	e	t	e	#
v	λ	e	λ	si

Input:	#	t	i	l	a	t	-	i	#
	1	2	3	4	5	6	7	8	9
Output:						s			
	1′	2′	3′	4′	5′	6′	7′	8′	9′

$$\mathrm{s}_o(x) =$$

$$\texttt{if } \underline{t}{-}i(x) \texttt{ then } \textsc{True} \texttt{ else if } \underline{t}e\#(x) \texttt{ then } \textsc{True} \texttt{ else } s(x)$$

Recursive version:

$$s_o(x) \quad = \quad \texttt{if } \underline{t} - i(x) \texttt{ then } \textsc{True} \texttt{ else}$$
$$\texttt{if } \underline{t}i(x) \texttt{ then } \textsc{False} \texttt{ else}$$
$$\texttt{if } \underline{t}i_o(x) \texttt{ then } \textsc{True} \texttt{ else } s(x)$$

Or, if morpheme boundaries are deleted in the output:

$$s_o(x) \quad = \quad \texttt{if } \underline{t}i(x) \texttt{ then } \textsc{False} \texttt{ else}$$
$$\texttt{if } \underline{t}i_o(x) \texttt{ then } \textsc{True} \texttt{ else } s(x)$$

Thanks to Adam Jardine (p.c.) for pointing this out!