Syntax-Prosody in Optimality Theory

Jenny Bellik & Nick Kalivoda

Much recent work on the syntax-prosody interface (Truckenbrodt 1999; Selkirk 2011; Elfner 2012; Myrberg 2013; Ito & Mester 2013; among many others) has fruitfully employed violable constraints within the framework of Optimality Theory (OT; Prince & Smolensky 1993).

An OT system consists of two functions, GEN and CON. In practice, most research has focused on CON—developing a constraint set and ranking that successfully model the phenomenon of interest. GEN has typically received less attention, with only a small number of candidates that could reasonably be expected to be optima under some ranking being explicitly considered in the analysis. This is probably due to the fact that manually generating all the possible candidates is prohibitively time-consuming for input syntactic structures with even a few terminals. In addition, since both inputs and outputs in syntax prosody work take the form of trees, it is impossible to generate and evaluate them automatically using regular expressions (as in OTWorkplace), which are too low in the Chomskyan hierarchy of language types to represent trees of arbitrary size.

To address these problems, we have developed SPOT (Bellik, Bellik & Kalivoda 2017; accessible online here), a JavaScript application that automatically generates prosodic candidate sets, and automatically evaluates each constraint for each candidate. In this talk, we illustrate the exponential increase in the number of relevant prosodic output candidates as the number of terminals in the syntactic string increases, and demonstrate how to use SPOT's HTML interface to automatically generate these candidates. We also show how SPOT is able to quickly and automatically assign violations for these candidates.

We also show how SPOT can be used to refine CON. Once the full set of candidates is considered, the precise formulation of each constraint can have unintended consequences for the computation of optimality. Small variations in the algorithmic definitions of familiar constraints like EQUALSISTERS, BINARITY, and NONRECURSIVITY will be shown to make significantly different predictions.

Syntax Prosody in OT

Jenny Bellik & Nick Kalivoda

jbellik@ucsc.edu nkalivod@ucsc.edu

SPOT Workshop @ UCSC, 11/18/2017

Introduction

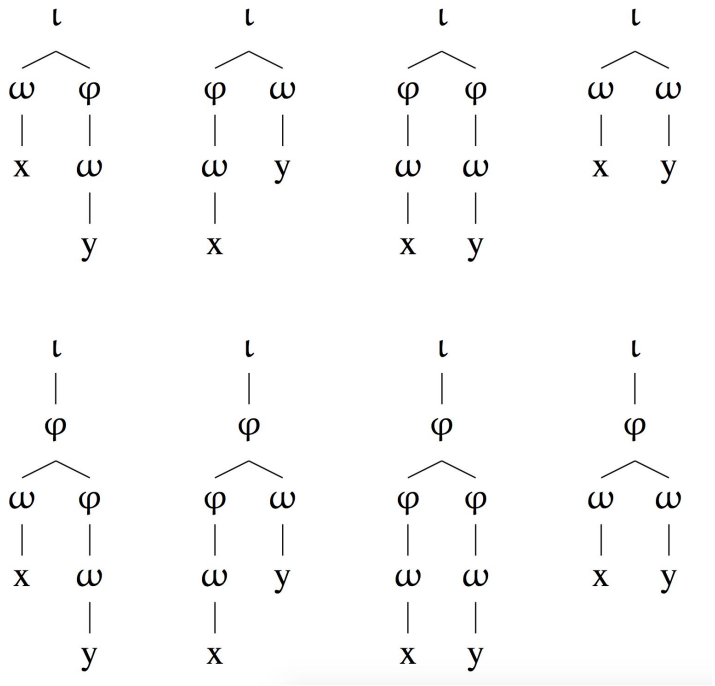
In Optimality Theory (Prince & Smolensky 1993), the winning output is supposed to be optimal

- amongst all the outputs of Gen
- as evaluated by Con.

Introduction

In syntax-prosody mapping:

- Both inputs and outputs are tree structures.
- As a consequence, outputs proliferate.



Introduction

This presentation:

- Introduce the SPOT application
- Demonstrate its utility
  - Exhaustive candidate generation and rapid evaluation
  - Theory comparison
  - Refinement of Con

SPOT application

SPOT is an open-source JavaScript application in development since 2014 that automates all three components of an OT system (Gen, Eval, Con).

- Application useable online without any programming: https://people.ucsc.edu/~jbellik/research/spot/interface1.html
- Source code available for download and modification: https://github.com/syntax-prosody-ot

Prosodic hierarchy in SPOT

ι $\leftarrow$ *Root node*

φ

ω $\leftarrow$ *Prosodic terminal*

xyz $\leftarrow$ *Lexical terminal*

Gen

In syntax-prosody work, inputs and outputs are both trees.

→ Gen needs to take a syntactic tree (s-tree) and build all the prosodic trees (p-trees) that could organize it

... under some set of assumptions:

- Strict Layering? (Selkirk 1984)
- Or some version of Weak Layering? (Ito & Mester 2003)

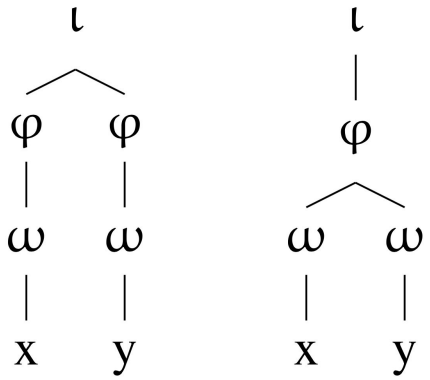
Gen with Strict Layering

(1) Gen(S-tree) = {set of all trees P such that P has the properties in (2)}

(2) Properties of P

- (a) P has a single root node ι.
- (b) Every non-root, non-terminal prosodic node is of category φ.
- (c) Every prosodic terminal node is of category ω.
- (d) Every lexical terminal is mapped onto a node of category ω.
- (e) The terminal string of P is identical to the terminal string of S.
- (f) The parent of every φ is of category ι. Non-recursivity
- (g) The parent of every ω is of category φ. Exhaustivity

$Gen_{[+NonRec,\ +Exh]}(S= [xy])$



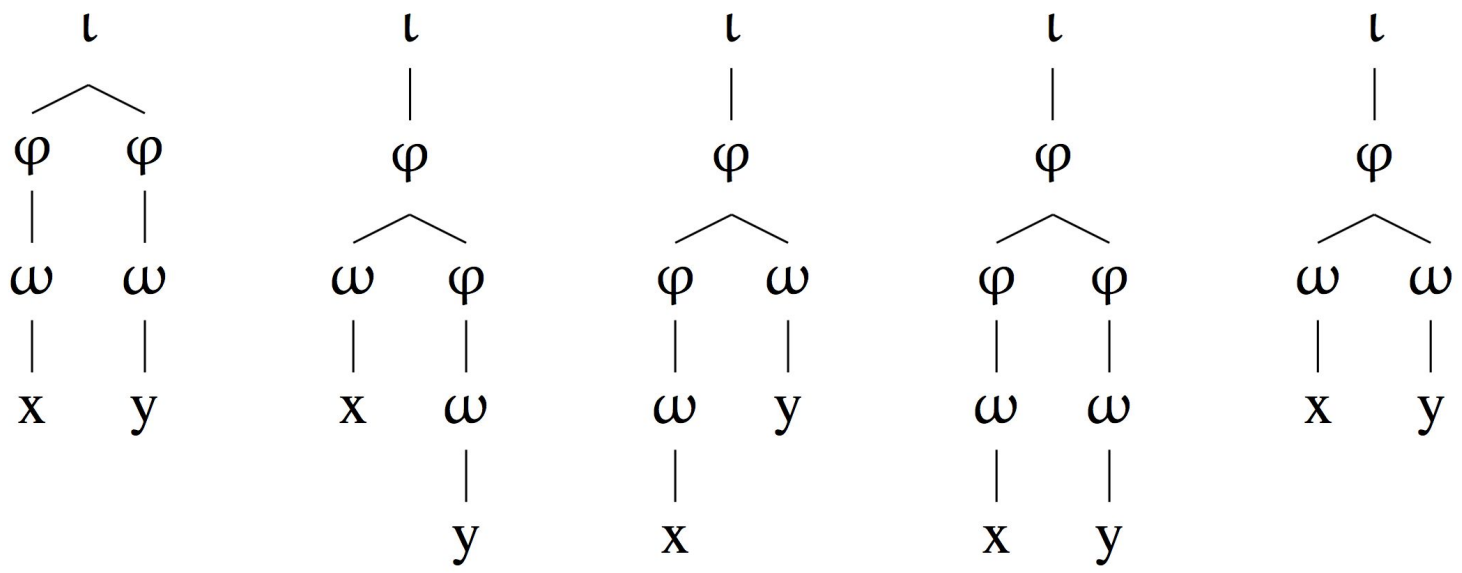
Gen with Weak Layering

In SPOT, the following settings of Gen can be turned on and off:

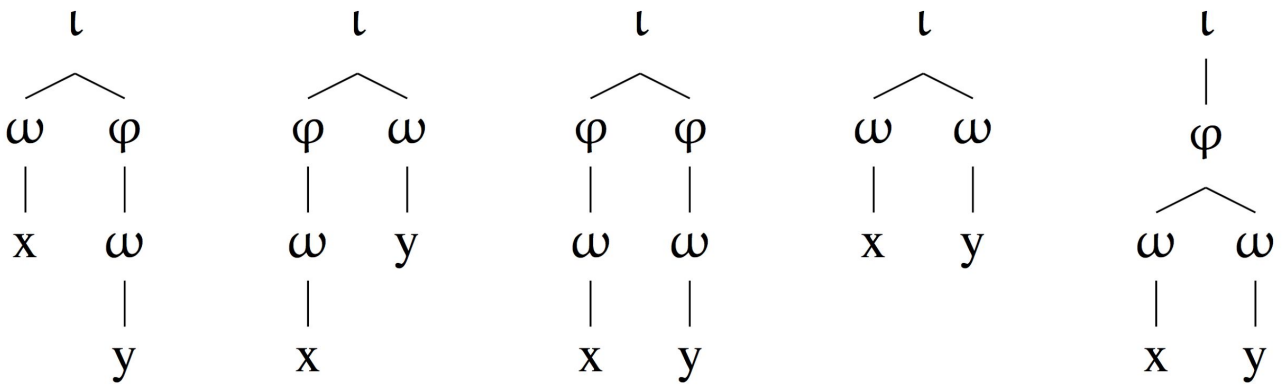
(1)	Headedness	*Every ι dominates at least one φ.*
(2)	Non-Recursivity	*No φ dominates a φ.*
(3)	Exhaustivity	*Every ω is dominated by some φ.*

Strict Layering Gen = [+Hd, + NonRec, + Exh]

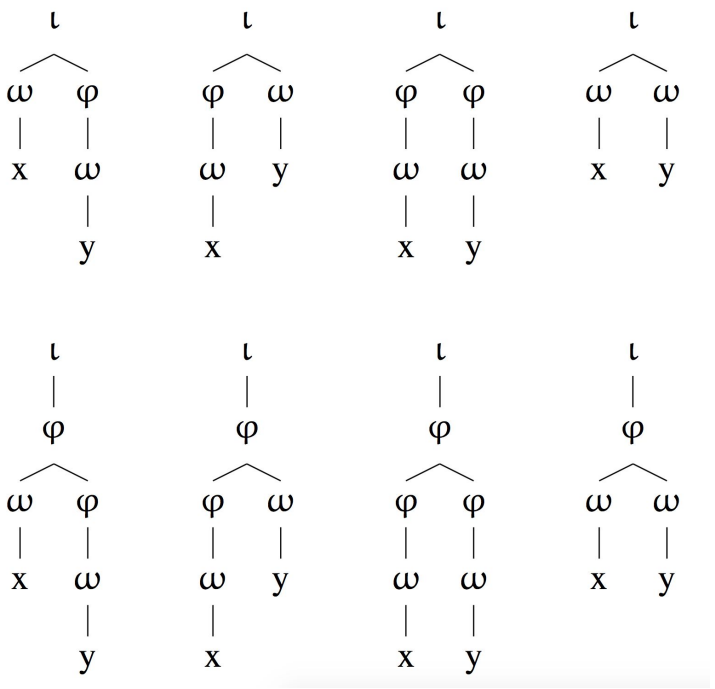
$Gen_{[-NonRec,\ +Exh]}(S= [xy])$



$Gen_{[+NonRec,\ -Exh,\ -Hd]}(S= [xy])$



Gen[−NonRec, −Exh, −Hd](S= [xy])



Comparing Gen functions

Each version of Gen results in rapid growth of the candidate set as the number of words increases.

By the time we get to six words:

- $\text{Gen}_{[-\text{NonRec},+\text{Exh},-\text{Hd}]}$ yields 18,853 parses.
- $\text{Gen}_{[-\text{NonRec},-\text{Exh},-\text{Hd}]}$ (weak layering) yields 25,216.

Using SPOT's Gen function

How can you get all these p-trees?

On SPOT, build a syntactic tree or just provide lexical terminals, then select the parameters for the Gen you are interested in.

Then click the Submit button and scroll down.

Con

SPOT implements many of the commonly used constraints:

- Mapping constraints:
  - Match-SP, Match-PS (Selkirk 2011)
  - Align-L, Align-R, Wrap (Truckenbrodt 1995, 1999)
- Markedness constraints:
  - NonRecursivity (Selkirk 1995; Truckenbrodt 1995, 1999)
  - EqualSisters (Myrberg 2012, 2013)
  - StrongStart (Selkirk 2011, Elfner 2012)
  - Japanese accent constraints NoLapse and AccentAsHead (Ito & Mester 2013)
- Theorists interested in other constraints can write them in JavaScript
  - or ask us to write them!

Eval

SPOT automatically evaluates all the prosodic candidate tree structures, ensuring rapid, accurate violation counting.

Demonstration

Refining Con

- Constraint definitions as typically formulated in the literature often become ambiguous when the full candidate set is taken into account
  - How many violations of, e.g., Binary does a ternary or 4-ary structure incur?
  - How many violations of NonRecursivity does a structure with multiple layers of recursion incur?
- JavaScript implementation of a constraint forces an unambiguous function definition.

Refining Con

- People already discuss what should count for S-P mapping constraints
  - Lexical categories only?
  - Phases? Spell-out domains? Extended projections?
  - Maximal XP only?
  - How to handle adjunction structures? (cf Bellik & Kalivoda 2015)
- But to really argue about this, we need to be able to distinguish the consequences of different mappings vs. different formulations of Markedness.
  - The remainder of this presentation explores the issue of how to formulate prosodic markedness constraints.

NonRecursivity

Truckenbrodt 1999 (p. 240):

(1) **NonRecursivity = NonRec-TB**
"Any two p-phrases that are not disjoint in extension are identical in extension."
I.e.: For every pair of φs a and b such that a dominates b, assign a violation for every ω dominated by a that is not also dominated by b

NonRecursivity

Other formulations are not hard to imagine:

(1) **NonRecursivity-children(p) = NonRec-children** (NonRec1 in SPOT)
Assign a violation for every node of category p that is immediately dominated by another node of category p

(2) **NonRecursivity-parent(p) = NonRec-parent** (not implemented in SPOT)
Assign a violation for every node of category p that dominates another node of category p. (Selkirk 1995)

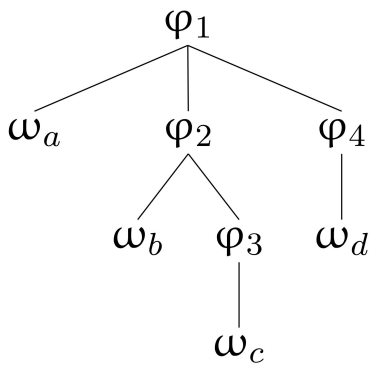
(3) **NonRecursivity-pairs(p) = NonRec-pairs** (not implemented in SPOT)
Assign a violation for every pair of nodes a and b such that a and b are both of category p and a dominates b.

NonRecursivity

	NonRec - TB	NonRec - children	NonRec - pairs	NonRec - parent
a. ((a b) c)	1	1	1	1
b. (((a b c)))	0	2	3	2
c. (a (b (c)))	3	2	3	2

- Each version of NonRecursivity partitions the candidate set differently.
- There is no stringency relationship among these constraint definitions
  - E.g. NonRec-TB assigns the fewest violations to (b) but the most to candidate (c)

NonRecursivity

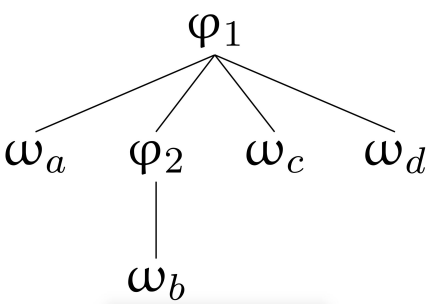
	NonRec - TB	NonRec - children	NonRec - pairs	NonRec - parent
	9 $(\varphi_1-\varphi_2) = 2$ $(\varphi_1-\varphi_3) = 3$ $(\varphi_1-\varphi_4) = 3$ $(\varphi_2-\varphi_3) = 1$	3 φ_2, φ_3, φ_4	4 (φ_1,φ_2), (φ_1, φ_3), (φ_1,φ_4), (φ_2,φ_3)	2 φ_1, φ_2

EqualSisters

Myrberg 2013, p. 75:

(1) **EqualSisters**
Sister nodes in prosodic structure are instantiations of the same prosodic category.

→ How do we implement this as a function?
How many violations for this structure?



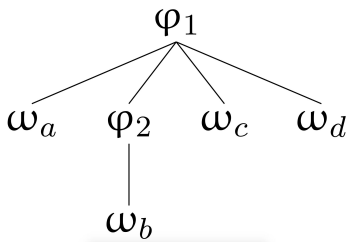
EqualSisters

Some possible versions of EqualSisters:

- **First privilege**: Assign a violation for every sister of a leftmost child node L whose prosodic category differs from that of L.
- **Pairwise**: Assign a violation for every pair of sisters whose prosodic categories differ.
- **Adjacent**: Assign a violation for every pair of adjacent sisters whose prosodic categories differ

EqualSisters

With super-binary branching structures, these three formulations can make three distinct predictions.

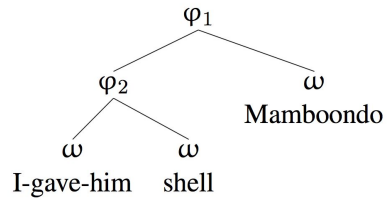
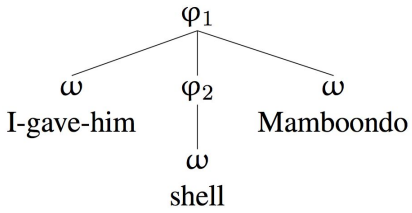
	ES-FirstPriv	ES-Adjacent	ES-Pairwise
	1 (ω_a,φ_2)	2 (ω_a,φ_2), (φ_2,ω_c)	3 (ω_a,φ_2), (φ_2,ω_c), (φ_2,ω_d),

Refining Con

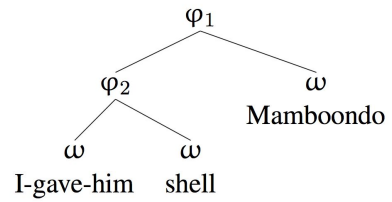
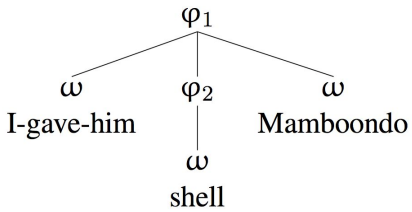
- **Why** do we need to differentiate the different formalizations of constraints like NonRecursivity, EqualSisters, etc.?
- **How** can we differentiate them?
- Let’s see an example: Kimatuumbi

Data from Odden 1987
Analysis from Truckenbrodt 1995, 1999

Consequences in Kimatuumbi

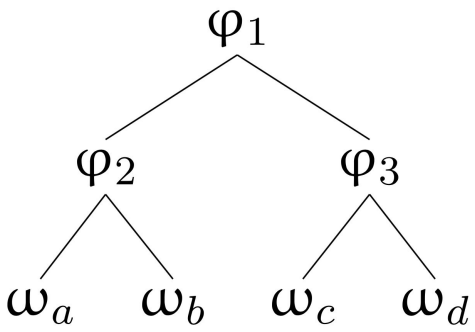
[$_{vP}$ I-gave-him [$_{NP}$ shell] [$_{NP}$ Mamboondo]]	**NonRec-TB**	**NonRec-children**
a. → 	1 $(\varphi_1 - \varphi_2)$	1
b. 	**W_2** $(\varphi_1 - \varphi_2)$	**e_1**

Consequences in Kimatuumbi

[$_{vP}$ I-gave-him [$_{NP}$ shell] [$_{NP}$ Mamboondo]]	**NonRec-TB** ⇆ **EqualSisters-Adj**	**NonRec-children** ⇆ **EqualSisters-1P**
a. → 	1	1
b. 	**W_2**	**e_1**

EqualSisters-Adj can do the same work as NonRec-TB.
Other formulations (e.g., EqualSisters-1stPrivilege) cannot.

Binarity

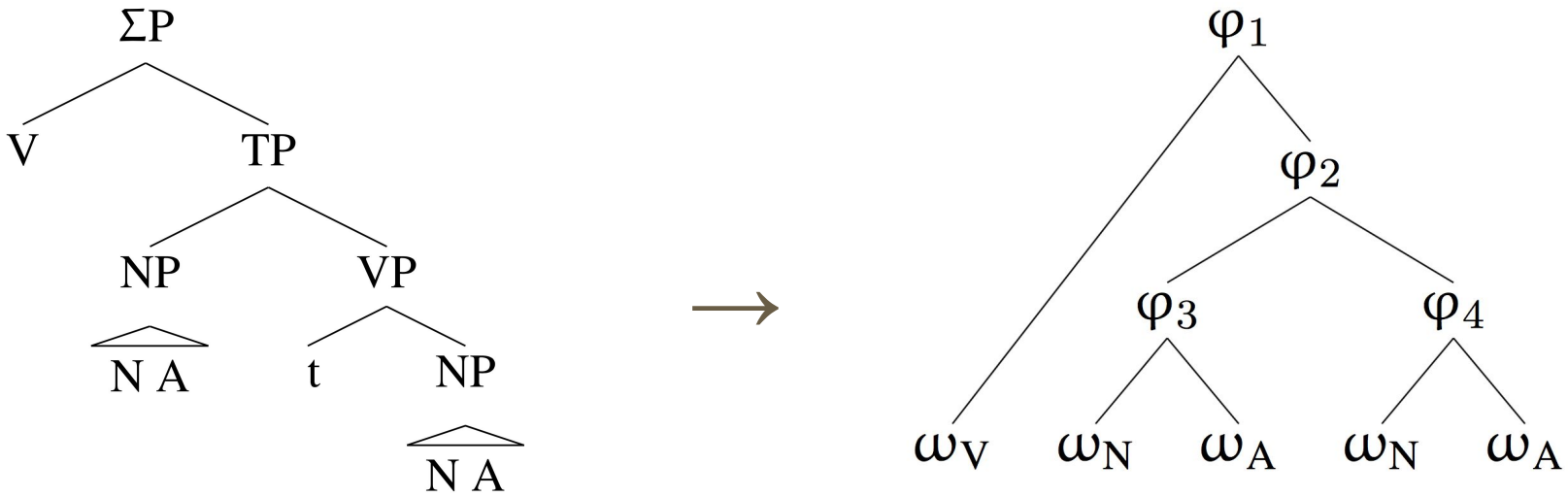


At least two hypotheses for binarity constraints have been proposed:

- **Branch-Counting Binarity:** a node should have exactly 2 branches. (Elfner 2012, Bellik & Kalivoda 2015)
- **Leaf-Counting Binarity:** a node should dominate exactly 2 terminals. (Ito & Mester 2007, to appear)

Branch-counting in Irish phrases

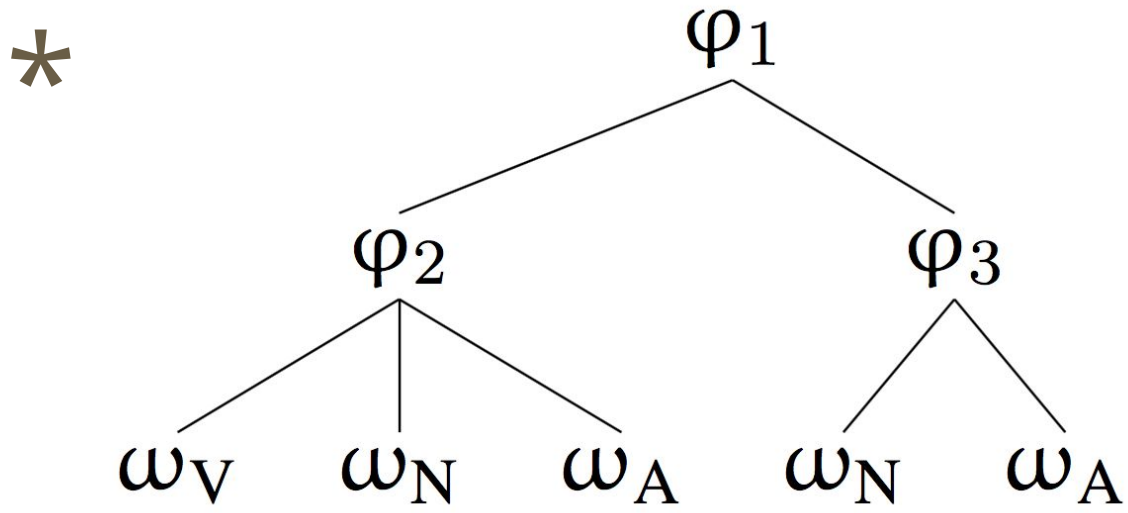
Elfner (2012) shows that in Irish, ω-adjunction to φ can be forced by maximal binarity.



The winning prosodic parse violates StrongStart (which Elfner shows to be active), but other candidates can be ruled out with BinMax.

Branch-counting in Irish phrases

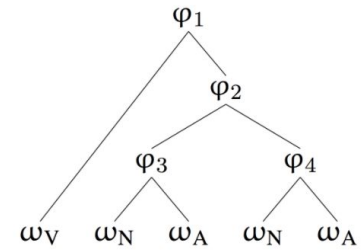
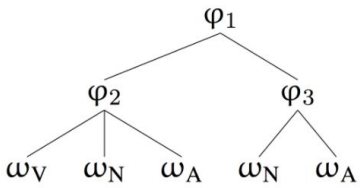
Specifically, Elfner (2012) needs to rule out the StrongStart-compliant candidate where V is in a φ with the first NP:



Branch-counting in Irish phrases

While Elfner's BinMax (=**BinMax-Br**), derives the right result...

...**BinMax-Leaves** would not work here as a substitute, whether gradient or categorical:

Input: [V [[N A] [[N A]]]]	**BinMax-Branches**	**Categorical BinMax-Leaves**	**Gradient BinMax-Leaves**
a. ▶ 	0	2 (φ_1, φ_2)	5 $(\varphi_1, \varphi_1, \varphi_1, \varphi_2, \varphi_2)$
b. 	**W_1** (φ_2)	e_2 (φ_1, φ_2)	**L_4** $(\varphi_1, \varphi_1, \varphi_1, \varphi_2)$

Leaf-counting in Japanese words

In Japanese, differences in prosodic size give rise to different compound prosodies (Ito & Mester 2007).

Japanese compound prosodies include:

- $(_{\omega}\ \omega\ (_{\omega}\ \omega\ \omega))$ *word*
- $(_{\varphi}\ \omega\ (_{\omega}\ \omega\ \omega))$ *mono-phrasal*
- $(_{\varphi}\ (_{\varphi}\ \omega)\ (_{\varphi}\ (_{\omega}\ \omega\ \omega)))$ *bi-phrasal*

Consequences for rendaku, junctural accent, and deaccenting.

Leaf-counting in Japanese words

Japanese compound prosody is sensitive to μ-count (Kubozono, Ito, & Mester 1997; Ito & Mester 2007):

- If the head ω is more than 4μ, a phrasal compound results.
- The 4μ template is also used in truncation (Ito 1990, Ito & Mester 1992) and language games (Tateishi 1989; Ito, Kitagawa, & Mester 1996)

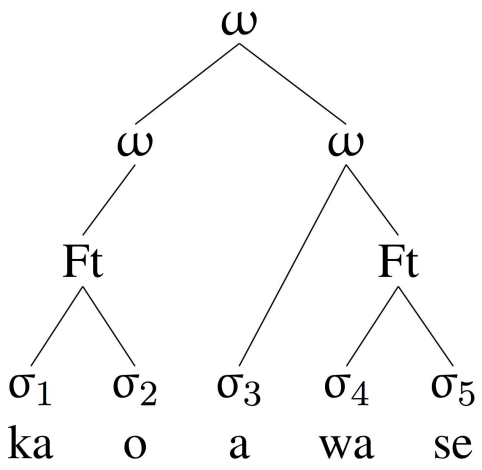
To deal with this, Ito & Mester (2007) invoke binarity constraints. Most relevant here:

BinMax(Head(ω)) "Heads of prosodic words are maximally binary."

Leaf-counting in Japanese words

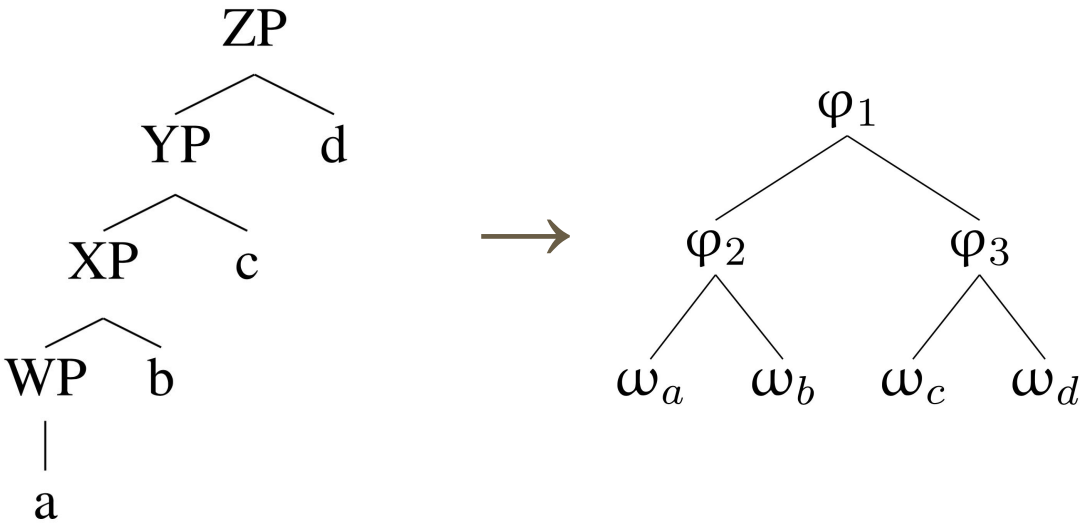
BinMax(Head(ω)) prevents the structure below for 5μ *kao-awase* from being the head of a ω → If *kao-awase* is the head, the compound has to be phrasal.

Crucially, ω^{Max} only violates binarity if we count leaves, rather than branches.



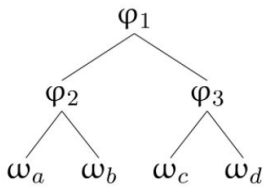
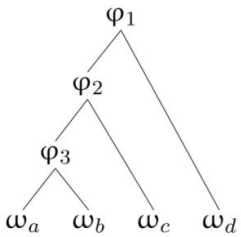
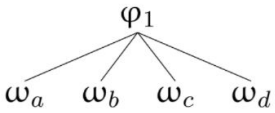
Branch-counting in Japanese phrases

At the phrase level in Japanese, we see a different type of binarity effect (Kubozono 1988; Ito & Mester 2013; Ishihara 2014):



Ito & Mester (to appear) appeal to **both** types of Binarity.

Branches *and* leaves?

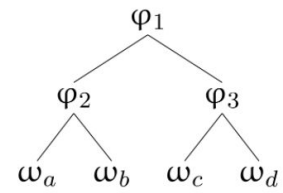
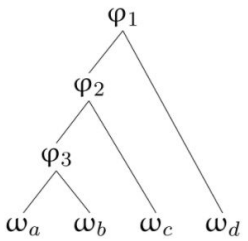
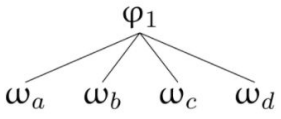
Input: [[[[a] b] c] d]	BinMax-Branches	BinMax-Leaves
a. 	0	1 (φ_1)
b. 	e_0	**W_2** (φ_1, φ_2)
c. 	**W_1** (φ1)	e_1 (φ_1)

BinMax-Leaves is needed for (a) to beat (b).

BinMax-Br is needed for (a) to beat (c).

Other constraints (not shown here) are either irrelevant or too low-ranked to make the distinction.

Replacing BinMax-Leaves

Input: [[[[a] b] c] d]	BinMax-Branches	BinMax-Leaves ⇆ NonRec-Parent	NonRec-Children
a. 	0	1 (φ_1)	2 (φ_2, φ_3)
b. 	e_0	$\mathbf{W_2}$ (φ_1, φ_2)	e_2 (φ_2, φ_3)
c. 	$\mathbf{W_1}$ (φ1)	$\mathbf{L_0}$	$\mathbf{L_0}$

Binarity

Evidence so far:

- Branch-counting is needed below φ (in Irish and Japanese)
- Leaf-counting is needed below ω (in Japanese)

Binarity in Danish Compounds

Ito & Mester (2015) discuss the role of prosodic structure in distribution of the Danish glottal accent, **stød** (here shown as ˀ).

(1) *Stød lost on short W1*

a.	/ruːˀg + brøːˀd/	→	[ruːg + brøːˀd]	'rye bread'
b.	/toːˀg + passageːˀr/	→	[toːg + passageːˀr]	'train passenger'

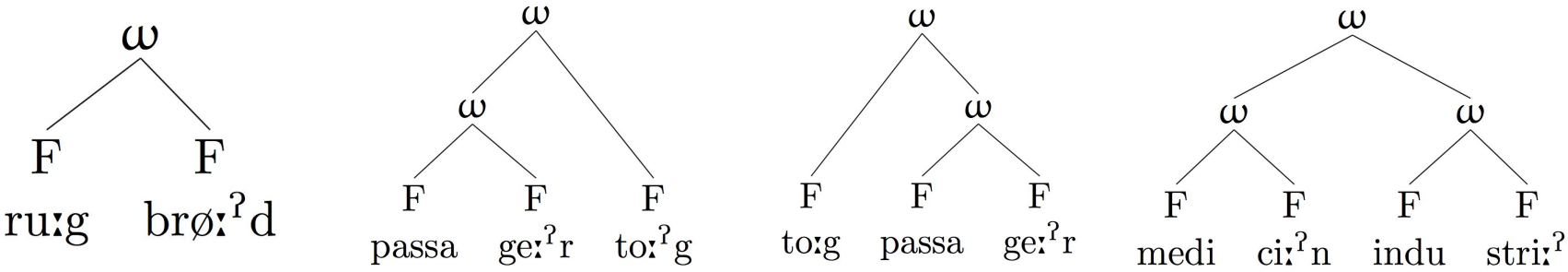
(2) *Stød retained on long W1*

a.	/passageːˀr + toːˀg/	→	[passageːˀr + toːˀg]	'passenger train'
b.	/medicːˀn + industriːˀ/	→	[medicːˀn + industriːˀ]	'medicine industry'

Binarity in Danish Compounds

Ito & Mester (2015) take stød to diagnose the right edge of ω.

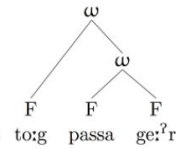
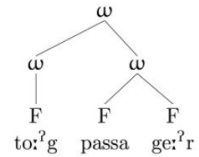
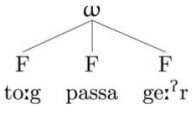
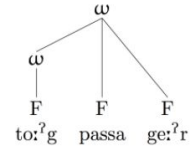
This allows us to make sense of the compounds, if they have different structures as follows:



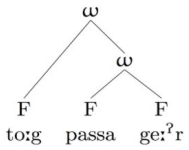
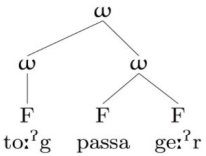
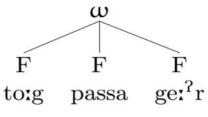
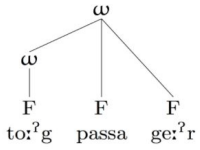
We (Bellik & Kalivoda 2017) propose an OT analysis using **Match(X^0,ω)**, **NonRecursivity-Children**, and **BinMax(ω,Branches)**.

Short-Long Compound

S+L → [$_\omega$ S + [$_\omega$ L$^?$]]

to:$^?$g + passage:$^?$r 'train passenger'	BinMax-Br	NonRec	Match(X^0,ω)
a. ▶ 	0	1	1
b. 	e_0	**W_2**	**L_0**
c. 	**W_1**	**L_0**	**W_2**
d. 	**W_1**	e_1	e_1

Inadequacy of Leaf-Counting Binarity

to:ˀg + passage:ˀr 'train passenger'	BinMax-Leaves	NonRec	Match(X^0,ω)
a. ▶ 	1	1	1
b. 	e_1	W_2	L_0
c. 	e_1	L_0	W_2
d. 	e_1	e_1	e_1

Leaf-counting **BinMax(ω,Leaves)** cannot replace **BinMax(ω,Branches)**.

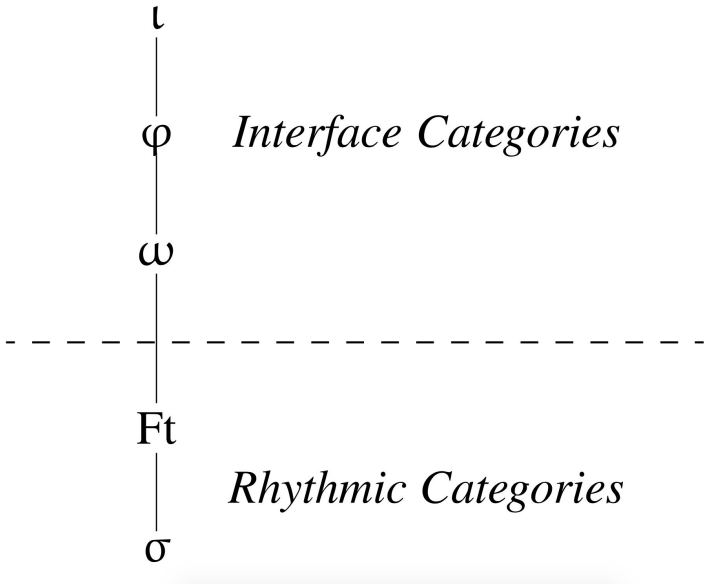
Binarity

Updated evidence:

- Branch-counting is needed below φ (in Irish and Japanese)
- ... and below ω (in Danish)
- Leaf-counting is needed below ω (in Japanese)

Hypothesis: Interface vs. rhythmic binarity

- Leaf-counting binarity only counts rhythmic categories.
- Branch-counting binarity can count any category.



Next: Allowing leaf-counting of interface categories may have undesirable typological results.

Binarity in Kinyambo phrasing

In Kinyambo, High Tone Deletion (1) diagnoses the φ-boundaries in (2).

(1) High Tone Deletion

$H \rightarrow \varnothing \ / \ (_{\varphi} \ldots (_{\omega} \ldots \_\_ \ldots) (_{\omega} \ldots H \ldots) \ldots)$

(2) a. $(_{\varphi}$ abak<u>o</u>zi b<u>á</u>kaj<u>ú</u>na)
'the workers helped'

b. $(_{\varphi}$ abak<u>o</u>zi bak<u>ú</u>ru) $(_{\varphi}$ b<u>á</u>kaj<u>ú</u>na)
'the mature workers helped'

(Bickmore 1990)

Binarity in Kinyambo phrasing

Bellik and Kalivoda (2015) used SPOT and OTW to model Kinyambo phrasing in 12 different OT systems.

- The formulation of Binarity did not affect whether a system successfully predicted Kinyambo phrasing as part of the typology.
  - Match systems predict Kinyambo phrasing with either version of binarity.
  - Align systems did not predict Kinyambo phrasing regardless of the type of binarity.
- However, the different formulations of Binarity affect the size of the typology.

In addition to the syntax-prosody mapping constraints and the binarity constraints, all systems included EqualSisters.

Binarity in Kinyambo phrasing

Systems using leaf-counting Binarity generated larger typologies than those using branch-counting Binarity (Bellik & Kalivoda 2015).

Typology sizes *(number of languages generated)*

	Match	Align
Binarity-Branches	11	12
Binarity-Leaves	15	28

The difference is larger in the Align systems than the Match systems.

This doesn't reveal much on its own, but supports the idea that for phrases, leaf-counting binarity might be undesirable, as well as unnecessary.

Conclusion

We have presented SPOT, an application designed to facilitate:

- Consideration of all prosodic candidates
  - Candidate generation
  - Violation assessment
- Refinements to Con
  - by enabling comparison of the consequences of different formulations of the same constraint idea
  - E.g., NonRecursivity, EqualSisters, Binarity
- Theory comparison
  - Easily assess violations of Align vs. Match, and compare their predictions.

Conclusion

We showed that small differences in constraint definitions can have significant consequences

- for the analysis of a given language
- for the number of constraints that are necessary
- for the predicted typology

Future directions

- Further comparison of different versions of standard constraints, as well as their interactions
- Expansion of SPOT
  - More prosodic and syntactic categories
  - Recursion at multiple levels
  - [Your suggestion here]

Thank you!

Works Cited

Bane, Max & Jason Riggle. 2012. Consequences of candidate omission. *Linguistic Inquiry* 43(4). 695-706.

Bellik, Jenny, Ozan Bellik & Nick Kalivoda. 2017. SPOT. JavaScript application. https://github.com/syntax-prosody-ot

Bellik, Jenny & Nick Kalivoda. 2015. Adjunction and Branchingness Effects in Match Theory. Poster presented at the Annual Meeting on Phonology.

—. 2017. Danish stød in recursive prosodic words. Poster presented at Northwestern Phon{etics, ology} 3.

Elfner, Emily. 2012. *Syntax-Prosody Interactions in Irish.* UMass Amherst dissertation.

Ito, Junko, and Armin Mester. Match Theory and Prosodic Wellformedness Constraints. To appear in Zhang, Hongming. ed. 2017. Proceedings of Tianjin Prosody Conference. Routledge.

Myrberg, Sara. 2010. *The intonational phonology of Stockholm Swedish*. Department of Scandinavian Languages, Stockholm University.

Myrberg, Sara. 2013. Sisterhood in prosodic branching. *Phonology* 30(1). 73-124.

Nespor, Marina & Irene Vogel. 1986. *Prosodic phonology*. Foris Publications.

Odden, David. 1987. Kimatuumbi Phrasal Phonology. *Phonology Yearbook* 4. 13-36.

Prince, Alan, Bruce Tesar, & Nazarré Merchant. 2017. OTWorkplace. Excel application. https://sites.google.com/site/otworkplace/home

Selkirk, Elizabeth. 1984. *Phonology and Syntax: The Relation between Sound and Structure*. Cambridge, MA: MIT Press.

—1986. On derived domains in sentence phonology. *Phonology* 3:371-405.

—1995. Sentence prosody: intonation, stress and phrasing. In *The Handbook of Phonological Theory*, ed. J. Goldsmith. London: Blackwell.

— 2011. The syntax-phonology interface. In J. Goldsmith, J. Riggle, & A.C.L. Yu (eds.), *The Handbook of phonological theory*, 435-484. London: Backwell.

Tateishi, Koichi. 1989. Theoretical implications of the Japanese musician's language. In *Proceedings of WCCFL 8*, eds. E. J. Fee & K. Hunt. Stanford: Stanford Linguistic Association, 384-398.

Truckenbrodt, Hubert. 1995. *Phonological phrases: Their relation to syntax, focus, and prominence.* MIT dissertation.

Truckenbrodt, Hubert. 1999. On the relation between syntactic phrases and phonological phrases. *Linguistic Inquiry* 30(2). 219-255.

Appendix: Gen with recursion

(1) Gen(S-tree) = {set of all trees P such that P has the properties in (2)}

(2) Properties of P

- (a) P has a single root node ι.
- (b) Every non-root, non-terminal prosodic node is of category φ.
- (c) Every prosodic terminal node is of category ω.
- (d) Every lexical terminal is mapped onto a node of category ω.
- (e) The terminal string of P is identical to the terminal string of S.
- ~~(f) The parent of every φ is of category ι.~~ Allow recursion
- (g) The parent of every ω is of category φ. Exhaustivity
- (h) Two nodes of the same category must dominate non-identical terminal strings No vacuous recursion

Appendix: Gen with recursion & non-exhaustive parsing

(1) Gen(S-tree) = {set of all trees P such that P has the properties in (2)}

(2) Properties of P

(a) P has a single root node ι.

(b) Every non-root, non-terminal prosodic node is of category φ.

(c) Every prosodic terminal node is of category ω.

(d) The terminal string of P is identical to the terminal string of S.

~~(e) The parent of every φ is of category ι.~~ ~~Non-recursivity~~

~~(f) The parent of every ω is of category φ.~~ ~~Exhaustivity~~

(g) P contains at least one node of category φ Headedness

Appendix: Gen with non-exhaustive parsing

(1) Gen(S-tree) = {set of all trees P such that P has the properties in (2)}

(2) Properties of P

(a) P has a single root node ι.

(b) Every non-root, non-terminal prosodic node is of category φ.

(c) Every prosodic terminal node is of category ω.

(d) Every lexical terminal is mapped onto a node of category ω.

(e) The terminal string of P is identical to the terminal string of S.

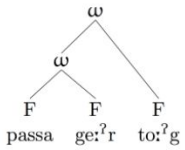
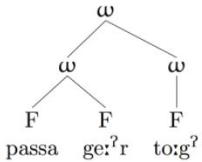
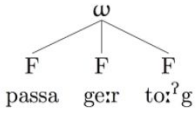
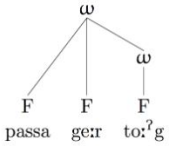
(f) The parent of every φ is of category ι. Non-recursivity

~~(g) The parent of every ω is of category φ.~~ ~~Exhaustivity~~

(h) P contains at least one node of category φ Headedness

Long-Short compound

L+S → $[_{\omega}\ [_{\omega}\ L^{?}] + {}_{\omega}\ S^{?}]$

passageɪ̯ʔr + toɪ̯ʔg 'passenger train'	BinMax-Br	NonRec	Match(X^0,ω)
a. ▶ 	0	1	1
b. 	e_0	**W_2**	**L_0**
c. 	**W_1**	**L_0**	**W_2**
d. 	**W_1**	e_1	e_1