Phonological Phrasing in Japanese

Nick Kalivoda*
SPOT at LSA 2021

*Based on Bellik, Ito, Kalivoda, & Mester (to appear 2021)

Japanese Mismatch

Kubozono (1989) found that a left-branching 4-word XP in Japanese maps to mismatching prosody:

(1) [[[Naomi-no] ane-to] yunomi-no] iro]
Naomi-GEN sister-GEN teacup-GEN color
'the color of the teacup of Naomi's sister'

→ ($_{\varphi}$ ($_{\varphi}$ Naomi-no ane-no) ($_{\varphi}$ yunomi-no iro))

Japanese Mismatch

Kubozono (1989) found that a left-branching 4-word XP in Japanese maps to mismatching prosody:

(1) [[[Naomi-no] ane-to] **yunomi-no] iro**]

Naomi-GEN sister-GEN teacup-GEN color

'the color of the teacup of Naomi's sister'

→ ($_{\varphi}$ ($_{\varphi}$ Naomi-no ane-no) ($_{\varphi}$ **yunomi-no iro**))

not a syntactic constituent

Japanese Mismatch

Kubozono (1989) found that a left-branching 4-word XP in Japanese maps to mismatching prosody:

(1) [[[Naomi-no] ane-to] **yunomi-no] iro**]
Naomi-GEN sister-GEN teacup-GEN color
'the color of the teacup of Naomi's sister'

→ $(_{\varphi}$ $(_{\varphi}$ Naomi-no ane-no) $(_{\varphi}$ **yunomi-no iro**))
not a syntactic constituent

Evidence: φ-initial rise on ω_1, ω_3
Second rise is downstepped due to φ^{Max} over $(_{\varphi}\ \omega_1\ \omega_2)$ and $(_{\varphi}\ \omega_3\ \omega_4)$

Japanese Matches

However, 4-word XPs of all other shapes undergo perfect matching (Kubozono 1989):

Right-branching:
[A [B [C D]]] → (A (B (C D)))

Balanced:
[[A B] [C D]] → ((A B) (C D))

Mixed (Left/Right):
[[A [B C]] D] → ((A (B C)) D)

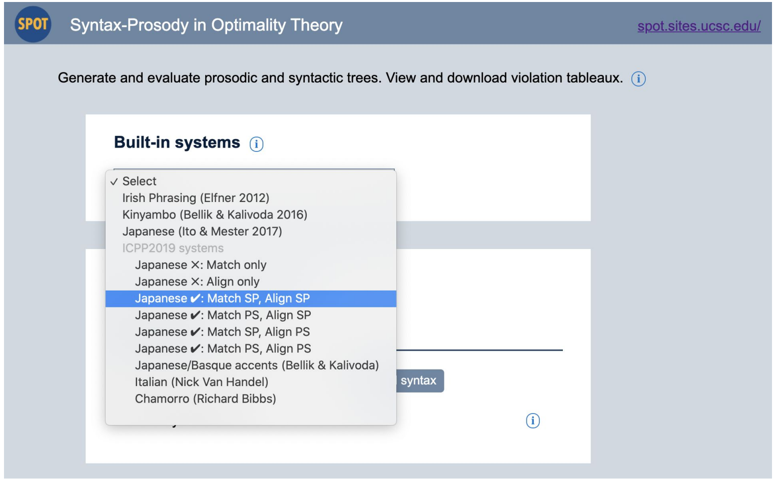
Mixed (Right/Left):
[A [[B C] D]] → (A ((B C) D))

Previous analyses (Selkirk 2011, Ishihara 2014, Kalivoda 2018) have attempted to analyze the left-branching mismatch in Match Theory (Selkirk 2011), but have not considered the matching cases.

We show that we need **Match and Align** to account for all these cases.

Studying *OT systems*

- An OT system $\boldsymbol{S} = (\mathbf{Gen}_S, \mathbf{Con}_S)$
- We define OT systems by using **SPOT** (Bellik et al. 2015-2020) and **OTWorkplace** (Prince et al. 2007-2020).
- The systems discussed in this talk are on the SPOT interface (linked from http://spot.sites.ucsc.edu):



Naming schema for our systems

S 'system'

Naming schema for our systems

S	'system'
Msp	Match(XP,φ) in CON
Mps	Match(φ,XP) in CON

Naming schema for our systems

S	'system'
Msp	Match(XP,φ) in CON
Mps	Match(φ,XP) in CON
Asp	Align(XP,L,φ,L) and Align(XP,R,φ,R) in CON
Aps	Align(φ,L,XP,L) and Align(φ,R,XP,R) in CON

Naming schema for our systems

S	'system'
Msp	Match(XP,φ) in CON
Mps	Match(φ,XP) in CON
Asp	Align(XP,L,φ,L) and Align(XP,R,φ,R) in CON
Aps	Align(φ,L,XP,L) and Align(φ,R,XP,R) in CON

Only CON varies; GEN constant across systems.

S.Msp.Asp

Matching and Alignment

GEN.Msp.Asp: Inputs

A candidate is an input-output pair.

GEN.Msp.Asp: Inputs

A candidate is an input-output pair.

(1) *Inputs*

Syntactic trees with 3 or 4 terminal nodes, where:

GEN.Msp.Asp: Inputs

A candidate is an input-output pair.

(1) *Inputs*

Syntactic trees with 3 or 4 terminal nodes, where:

- every non-terminal node is a binary-branching XP

GEN.Msp.Asp: Inputs

A candidate is an input-output pair.

(1) *Inputs*

Syntactic trees with 3 or 4 terminal nodes, where:

- every non-terminal node is a binary-branching XP
- every terminal node is an X^0

GEN.Msp.Asp: Inputs

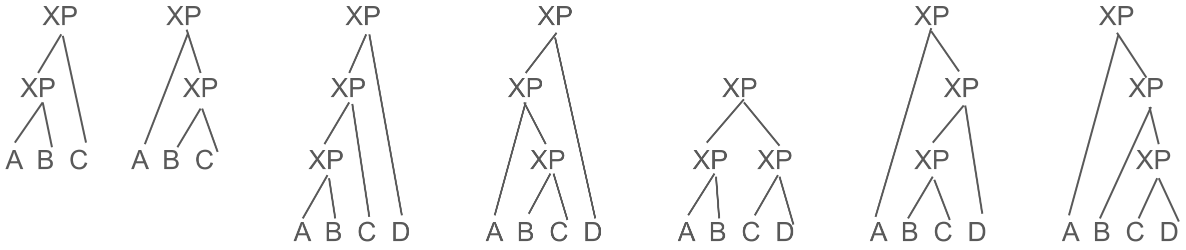
A candidate is an input-output pair.

(1) *Inputs*

Syntactic trees with 3 or 4 terminal nodes, where:

- every non-terminal node is a binary-branching XP
- every terminal node is an X^0

I.e.:



GEN.Msp.Asp: Outputs

(1) *Outputs*

For a syntactic input sTree, every prosodic tree pTree such that:

GEN.Msp.Asp: Outputs

(1) *Outputs*

For a syntactic input sTree, every prosodic tree pTree such that:

- non-terminal nodes are of category φ

GEN.Msp.Asp: Outputs

(1) *Outputs*

For a syntactic input sTree, every prosodic tree pTree such that:

- non-terminal nodes are of category φ
- terminal nodes are of category ω

GEN.Msp.Asp: Outputs

(1) *Outputs*

For a syntactic input sTree, every prosodic tree pTree such that:

- non-terminal nodes are of category φ
- terminal nodes are of category ω
- the terminal nodes in sTree stand in a one-to-one correspondence relation with the terminal nodes in pTree, with linear order preserved.

CON.Msp.Asp

(1) *Mapping constraints*

(a) **MATCH(XP,φ)**
Assign one violation for every node of category XP in the syntactic tree such that there is no node of category φ in the prosodic tree that dominates all and only the same terminal nodes as XP.

Con.Msp.Asp

(1) *Mapping constraints*

(a) Match(XP,φ)

Assign one violation for every node of category XP in the syntactic tree such that there is no node of category φ in the prosodic tree that dominates all and only the same terminal nodes as XP.

(b) AlignL(XP,φ)

Assign one violation for every node of category XP in the syntactic tree whose left edge is not aligned with the left edge of a node of category φ in the prosodic tree.

CON.Msp.Asp

(1) *Mapping constraints*

(a) **MATCH(XP,φ)**
Assign one violation for every node of category XP in the syntactic tree such that there is no node of category φ in the prosodic tree that dominates all and only the same terminal nodes as XP.

(b) **ALIGNL(XP,φ)**
Assign one violation for every node of category XP in the syntactic tree whose left edge is not aligned with the left edge of a node of category φ in the prosodic tree.

(c) **ALIGNR(XP,φ)**
Assign one violation for every node of category XP in the syntactic tree whose right edge is not aligned with the right edge of a node of category φ in the prosodic tree.

CON.Msp.Asp

(1) *Mapping constraints*

- (a) MATCH(XP,φ)
- (b) ALIGNL(XP,φ)
- (c) ALIGNR(XP,φ)

(2) *Markedness constraints*

- **(a) BinMin(φ,ω)**

  Assign one violation for every node of category φ in the prosodic tree that contains fewer than two nodes of category ω.

CON.Msp.Asp

(1) *Mapping constraints*

(a) MATCH(XP,φ)

(b) ALIGNL(XP,φ)

(c) ALIGNR(XP,φ)

(2) *Markedness constraints*

(a) BinMin(φ,ω)

Assign one violation for every node of category φ in the prosodic tree that contains fewer than two nodes of category ω.

(b) **BinMax(φ,ω)**

Assign one violation for every node of category φ in the prosodic tree that dominates more than two nodes of category ω.

CON.Msp.Asp

(1) *Mapping constraints*

- (a) MATCH(XP,φ)
- (b) ALIGNL(XP,φ)
- (c) ALIGNR(XP,φ)

(2) *Markedness constraints*

- **(a) BinMin(φ,ω)**
  Assign one violation for every node of category φ in the prosodic tree that contains fewer than two nodes of category ω.
- (b) **BinMax(φ,ω)**
  Assign one violation for every node of category φ in the prosodic tree that dominates more than two nodes of category ω.
- (c) **BinMax(φ,branches)**
  Assign one violation for every node of category φ in the prosodic tree that has more than two children.

Factorial Typology of S.Msp.Asp

	[[[A B] C] D]	**[[A [B C]] D]**	**[A [[B C] D]]**	**[A [B [C D]]]**
L.1	((A B) (C D))	((A B) (C D))	((A B) (C D))	((A B) (C D))
L.2	((A B) ((C) D))	((A B) ((C) D))	((A B) ((C) D))	((A B) (C D))
L.3	(((A B) C) D)	((A (B C)) D)	(A ((B C) D))	((A B) (C D))
L.4	(((A B) C) D)	(A (B C) D)	(A (B C) D)	((A B) (C D))
L.5	((A B) (C D))	(A (B C) D)	(A (B C) D)	((A B) (C D))
L.6	((A B) ((C) D))	(A (B C) D)	(A (B C) D)	((A B) (C D))
L.7	((A B) (C D))	((A (B)) (C D))	((A (B)) (C D))	((A (B)) (C D))
L.8	((A B) ((C) D))	((A (B)) ((C) D))	((A (B)) ((C) D))	((A (B)) (C D))
L.9	((A B) (C D))	(A (B C) D)	(A (B C) D)	((A (B)) (C D))
L.10	((A B) ((C) D))	(A (B C) D)	(A (B C) D)	((A (B)) (C D))
L.11	(((A B) C) D)	((A (B C)) D)	(A ((B C) D))	(A (B (C D)))
L.12	**((A B) (C D))**	**((A (B C)) D)**	**(A ((B C) D))**	**(A (B (C D)))**
L.13	(((A B) C) D)	(A (B C) D)	(A (B C) D)	(A (B (C D)))
L.14	((A B) (C D))	(A (B C) D)	(A (B C) D)	(A (B (C D)))

Factorial Typology of S.Msp.Asp

	[[[A B] C] D]	**[[A [B C]] D]**	**[A [[B C] D]]**	**[A [B [C D]]]**
L.1	((A B) (C D))	((A B) (C D))	((A B) (C D))	((A B) (C D))
L.2	((A B) ((C) D))	((A B) ((C) D))	((A B) ((C) D))	((A B) (C D))
L.3	(((A B) C) D)	((A (B C)) D)	(A ((B C) D))	((A B) (C D))
L.4	(((A B) C) D)	(A (B C) D)	(A (B C) D)	((A B) (C D))
L.5	((A B) (C D))	(A (B C) D)	(A (B C) D)	((A B) (C D))
L.6	((A B) ((C) D))	(A (B C) D)	(A (B C) D)	((A B) (C D))
L.7	((A B) (C D))	((A (B)) (C D))	((A (B)) (C D))	((A (B)) (C D))
L.8	((A B) ((C) D))	((A (B)) ((C) D))	((A (B)) ((C) D))	((A (B)) (C D))
L.9	((A B) (C D))	(A (B C) D)	(A (B C) D)	((A (B)) (C D))
L.10	((A B) ((C) D))	(A (B C) D)	(A (B C) D)	((A (B)) (C D))
L.11	(((A B) C) D)	((A (B C)) D)	(A ((B C) D))	(A (B (C D)))
L.12	**((A B) (C D))**	**((A (B C)) D)**	**(A ((B C) D))**	**(A (B (C D)))**
L.13	(((A B) C) D)	(A (B C) D)	(A (B C) D)	(A (B (C D)))
L.14	((A B) (C D))	(A (B C) D)	(A (B C) D)	(A (B (C D)))

Not shown:
[[A B] C]
[A [B C]]
[[A B] [C D]]

These match in all 14 languages

Japanese pattern in S.Msp.Asp

	[[[A B] C] D]	[[A [B C]] D]	[A [[B C] D]]	[A [B [C D]]]
L.1	((A B) (C D))	((A B) (C D))	((A B) (C D))	((A B) (C D))
L.2	((A B) ((C) D))	((A B) ((C) D))	((A B) ((C) D))	((A B) (C D))
L.3	(((A B) C) D)	((A (B C)) D)	(A ((B C) D))	((A B) (C D))
L.4	(((A B) C) D)	(A (B C) D)	(A (B C) D)	((A B) (C D))
L.5	((A B) (C D))	(A (B C) D)	(A (B C) D)	((A B) (C D))
L.6	((A B) ((C) D))	(A (B C) D)	(A (B C) D)	((A B) (C D))
L.7	((A B) (C D))	((A (B)) (C D))	((A (B)) (C D))	((A (B)) (C D))
L.8	((A B) ((C) D))	((A (B)) ((C) D))	((A (B)) ((C) D))	((A (B)) (C D))
L.9	((A B) (C D))	(A (B C) D)	(A (B C) D)	((A (B)) (C D))
L.10	((A B) ((C) D))	(A (B C) D)	(A (B C) D)	((A (B)) (C D))
L.11	(((A B) C) D)	((A (B C)) D)	(A ((B C) D))	(A (B (C D)))
L.12	**((A B) (C D))**	**((A (B C)) D)**	**(A ((B C) D))**	**(A (B (C D)))**
L.13	(((A B) C) D)	(A (B C) D)	(A (B C) D)	(A (B (C D)))
L.14	((A B) (C D))	(A (B C) D)	(A (B C) D)	(A (B (C D)))

← Japanese

Japanese pattern in S.Msp.Asp

	[[[A B] C] D]	**[[A [B C]] D]**	**[A [[B C] D]]**	**[A [B [C D]]]**
L.1	((A B) (C D))	((A B) (C D))	((A B) (C D))	((A B) (C D))
L.2	((A B) ((C) D))	((A B) ((C) D))	((A B) ((C) D))	((A B) (C D))
L.3	(((A B) C) D)	((A (B C)) D)	(A ((B C) D))	((A B) (C D))
L.4	(((A B) C) D)	(A (B C) D)	(A (B C) D)	((A B) (C D))
L.5	((A B) (C D))	(A (B C) D)	(A (B C) D)	((A B) (C D))
L.6	((A B) ((C) D))	(A (B C) D)	(A (B C) D)	((A B) (C D))
L.7	((A B) (C D))	((A (B)) (C D))	((A (B)) (C D))	((A (B)) (C D))
L.8	((A B) ((C) D))	((A (B)) ((C) D))	((A (B)) ((C) D))	((A (B)) (C D))
L.9	((A B) (C D))	(A (B C) D)	(A (B C) D)	((A (B)) (C D))
L.10	((A B) ((C) D))	(A (B C	[illegible]	((A (B)) (C D))
L.11	(((A B) C) D)	((A (B C)) D)	(A ((B C) D))	(A (B (C D)))
L.12	**((A B) (C D))**	**((A (B C)) D)**	**(A ((B C) D))**	**(A (B (C D)))**
L.13	(((A B) C) D)	(A (B C) D)	(A (B C) D)	(A (B (C D)))
L.14	((A B) (C D))	(A (B C) D)	(A (B C) D)	(A (B (C D)))

Isomorphic mappings

← Japanese

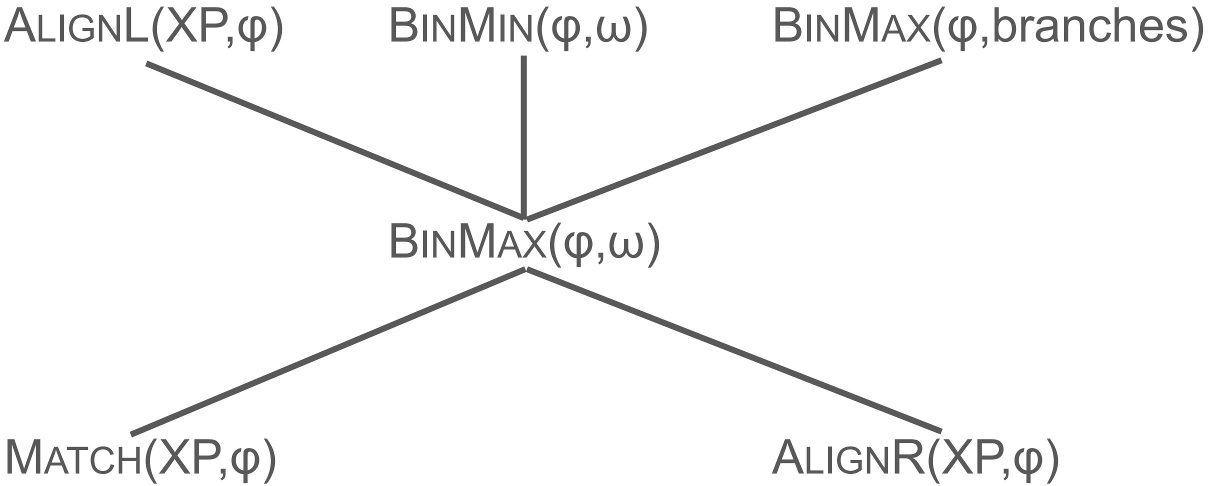
Japanese pattern in S.Msp.Asp

	[[[A B] C] D]	[[A [B C]] D]	[A [[B C] D]]	[A [B [C D]]]
L.1	((A B) (C D))	((A B) (C D))	((A B) (C D))	((A B) (C D))
L.2	((A B) ((C) D))	((A B) ((C) D))	((A B) ((C) D))	((A B) (C D))
L.3	(((A B) C) D)	((A (B C)) D)	(A ((B C) D))	((A B) (C D))
L.4	(((A B) C) D)	(A (B C) D)	(A (B C) D)	((A B) (C D))
L.5	((A B) (C D))	(A (B C) D)	(A (B C) D)	((A B) (C D))
L.6	((A B) ((C) D))	(A (B C) D)	(A (B C) D)	((A B) (C D))
L.7	((A B) (C D))	((A (B)) (C D))	((A (B)) (C D))	((A (B)) (C D))
L.8	((A B) ((C) D))	((A (B)) ((C) D))	((A (B)) ((C) D))	((A (B)) (C D))
L.9	((A B) (C D))	(A (B C) D)	(A (B C) D)	((A (B)) (C D))
L.10	[illegible]	(A (B C) D)	(A (B C) D)	((A (B)) (C D))
L.11	(((A B) C) D)	((A (B C)) D)	(A ((B C) D))	(A (B (C D)))
L.12	**((A B) (C D))**	**((A (B C)) D)**	**(A ((B C) D))**	**(A (B (C D)))**
L.13	(((A B) C) D)	(A (B C) D)	(A (B C) D)	(A (B (C D)))
L.14	((A B) (C D))	(A (B C) D)	(A (B C) D)	(A (B (C D)))

Rebracketing

← Japanese

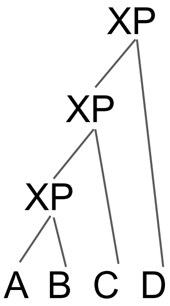
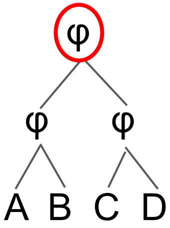
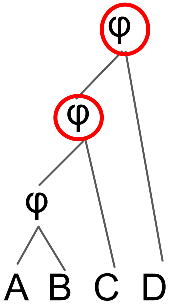
Grammar of L.12 in Sp.Msp.Asp



Support for L.12

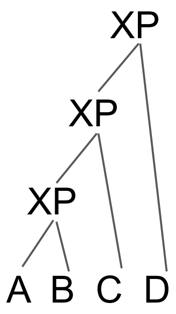
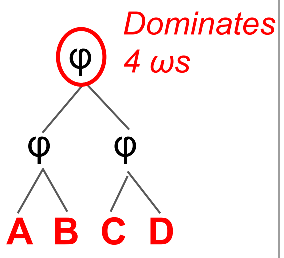
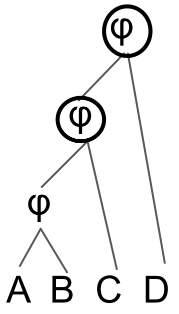
Input	Winner	Loser	AlignL (XP,φ)	BinMin (φ,ω)	BinMax (φ,br)	BinMax (φ,ω)	Match (XP,φ)	AlignR (XP,φ)
[A[B[CD]]]	(A(B(CD)))	((AB)(CD))	**W**			**L**	W	
[A[B[CD]]]	(A(B(CD)))	((A(B))(CD))		**W**		**L**	W	
[A[[BC]D]]	(A((BC)D))	(A(BC)D)			**W**	**L**	W	
[[[AB]C]D]	((AB)(CD))	(((AB)C)D)				**W**	**L**	**L**

Support for L.12: *L-branching→rebracketed*

Input	**Winner**	**Loser**	ALIGNL (XP,φ)	BINMIN (φ,ω)	BINMAX (φ,br)	**BINMAX (φ,ω)**	MATCH (XP,φ)	ALIGNR (XP,φ)
[A[B[CD]]]	(A(B(CD)))	((AB)(CD))	W			L	W	
[A[B[CD]]]	(A(B(CD)))	((A(B))(CD))		W		L	W	
[A[[BC]D]]	(A((BC)D))	(A(BC)D)			W	L	W	
						W	L	L

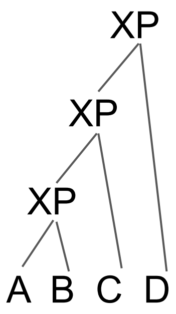
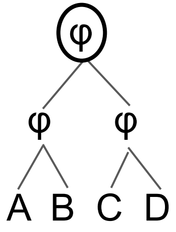
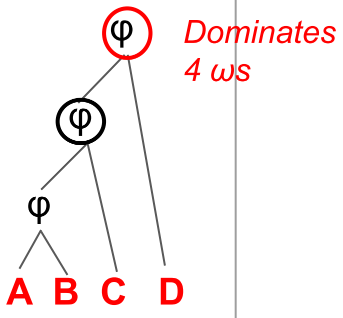
BINMAX(φ,ω) prefers winner.

Support for L.12: *L-branching→rebracketed*

Input	**Winner**	**Loser**	ALIGNL (XP,φ)	BINMIN (φ,ω)	BINMAX (φ,br)	**BINMAX (φ,ω)**	MATCH (XP,φ)	ALIGNR (XP,φ)
[A[B[CD]]]	(A(B(CD)))	((AB)(CD))	W			L	W	
[A[B[CD]]]	(A(B(CD)))	((A(B))(CD))		W		L	W	
[A[[BC]D]]	(A((BC)D))	(A(BC)D)			W	L	W	
						W	L	L

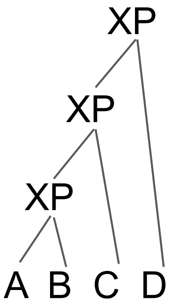
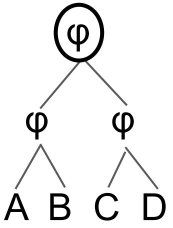
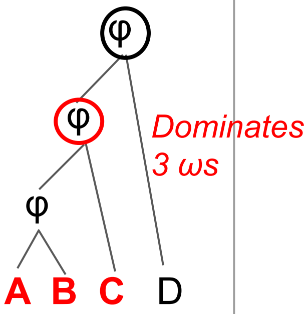
BINMAX(φ,ω) prefers winner.

Support for L.12: *L-branching→rebracketed*

Input	**Winner**	**Loser**	ALIGNL (XP,φ)	BINMIN (φ,ω)	BINMAX (φ,br)	**BINMAX (φ,ω)**	MATCH (XP,φ)	ALIGNR (XP,φ)
[A[B[CD]]]	(A(B(CD)))	((AB)(CD))	W			L	W	
[A[B[CD]]]	(A(B(CD)))	((A(B))(CD))		W		L	W	
[A[[BC]D]]	(A((BC)D))	(A(BC)D)			W	L	W	
						W	L	L

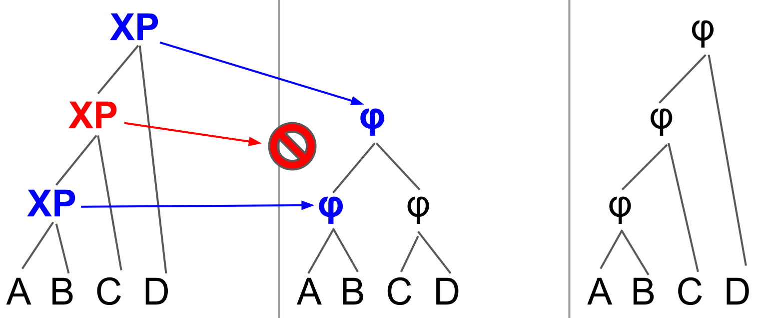
BINMAX(φ,ω) prefers winner.

Support for L.12: *L-branching→rebracketed*

Input	**Winner**	**Loser**	ALIGNL (XP,φ)	BINMIN (φ,ω)	BINMAX (φ,br)	**BINMAX (φ,ω)**	MATCH (XP,φ)	ALIGNR (XP,φ)
[A[B[CD]]]	(A(B(CD)))	((AB)(CD))	W			L	W	
[A[B[CD]]]	(A(B(CD)))	((A(B))(CD))		W		L	W	
[A[[BC]D]]	(A((BC)D))	(A(BC)D)			W	L	W	
						W	L	L

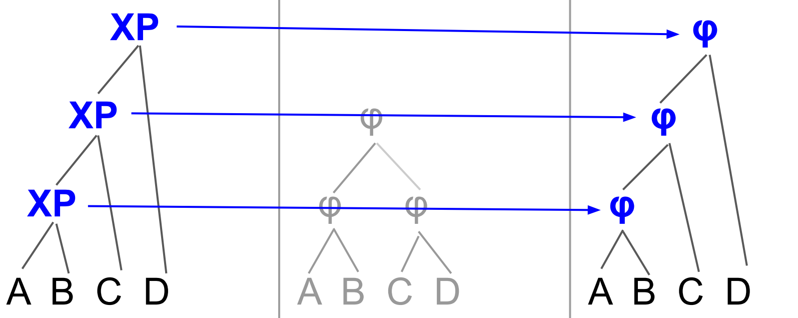
BINMAX(φ,ω) prefers winner.

Support for L.12: *L-branching→rebracketed*

Input	**Winner**	**Loser**	ALIGNL (XP,φ)	BINMIN (φ,ω)	BINMAX (φ,br)	BINMAX (φ,ω)	**MATCH (XP,φ)**	ALIGNR (XP,φ)
[A[B[CD]]]	(A(B(CD)))	((AB)(CD))	W			L	W	
[A[B[CD]]]	(A(B(CD)))	((A(B))(CD))		W		L	W	
[A[[BC]D]]	(A((BC)D))	(A(BC)D)			W	L	W	
						W	L	L

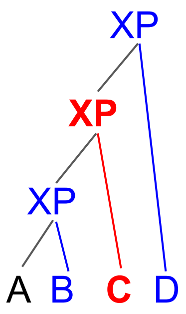
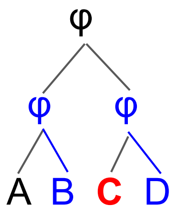
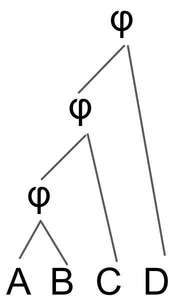
MATCH(XP,φ) prefers loser; **[$_{XP}$ ABC]** unmmatched in winner.

Support for L.12: *L-branching→rebracketed*

Input	**Winner**	**Loser**	ALIGNL (XP,φ)	BINMIN (φ,ω)	BINMAX (φ,br)	BINMAX (φ,ω)	**MATCH (XP,φ)**	ALIGNR (XP,φ)
[A[B[CD]]]	(A(B(CD)))	((AB)(CD))	W			L	W	
[A[B[CD]]]	(A(B(CD)))	((A(B))(CD))		W		L	W	
[A[[BC]D]]	(A((BC)D))	(A(BC)D)			W	L	W	
						W	L	L

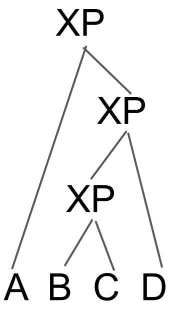
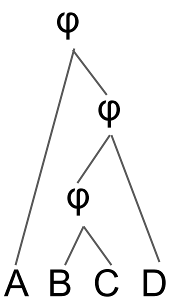
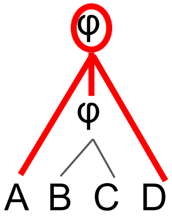
MATCH(XP,φ) prefers loser; $[_{XP}$ ABC] unmatched in winner.

Support for L.12: *L-branching→rebracketed*

Input	**Winner**	**Loser**	ALIGNL (XP,φ)	BINMIN (φ,ω)	BINMAX (φ,br)	BINMAX (φ,ω)	MATCH (XP,φ)	**ALIGNR (XP,φ)**
[A[B[CD]]]	(A(B(CD)))	((AB)(CD))	W			L	W	
[A[B[CD]]]	(A(B(CD)))	((A(B))(CD))		W		L	W	
[A[[BC]D]]	(A((BC)D))	(A(BC)D)			W	L	W	
						**W**	**L**	**L**

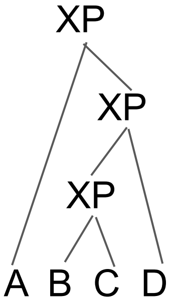
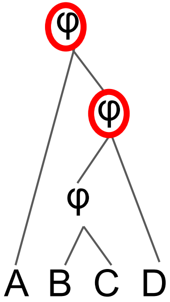
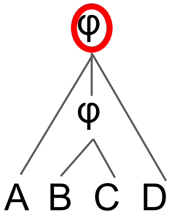
ALIGNR(XP,φ) prefers loser; **C]↛C)** in winner.

Support for L.12: *mixed-branching→isomorphic*

Input	**Winner**	**Loser**	ALIGNL (XP,φ)	BINMIN (φ,ω)	**BINMAX (φ,br)**	BINMAX (φ,ω)	MATCH (XP,φ)	ALIGNR (XP,φ)
[A[B[CD]]]	(A(B(CD)))	((AB)(CD))	W			L	W	
[A[B[CD]]]	(A(B(CD)))	((A(B))(CD))		W		L	W	
					W	L	W	
[[[AB]C]D]	((AB)(CD))	(((AB)C)D)				W	L	L

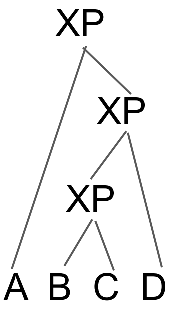
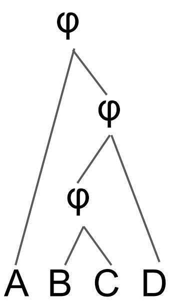
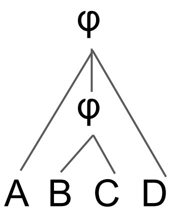
BINMAX(φ,branches) prefers winner; loser contains **ternary ($(_\varphi$ A φ D)**.

Support for L.12: *mixed-branching→isomorphic*

Input	**Winner**	**Loser**	ALIGNL (XP,φ)	BINMIN (φ,ω)	BINMAX (φ,br)	**BINMAX (φ,ω)**	MATCH (XP,φ)	ALIGNR (XP,φ)
[A[B[CD]]]	(A(B(CD)))	((AB)(CD))	W			L	W	
[A[B[CD]]]	(A(B(CD)))	((A(B))(CD))		W		L	W	
					W	**L**	W	
[[[AB]C]D]	((AB)(CD))	(((AB)C)D)				W	L	L

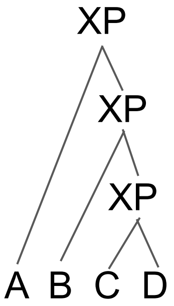
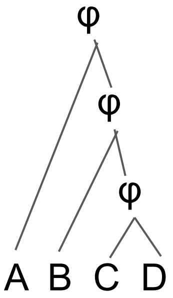
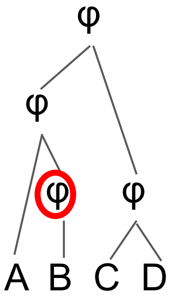
BINMAX(φ,ω) prefers loser.

Support for L.12: *mixed-branching→isomorphic*

Input	**Winner**	**Loser**	AlignL (XP,φ)	BinMin (φ,ω)	**BinMax (φ,br)**	**BinMax (φ,ω)**	**Match (XP,φ)**	AlignR (XP,φ)
[A[B[CD]]]	(A(B(CD)))	((AB)(CD))	W			L	W	
[A[B[CD]]]	(A(B(CD)))	((A(B))(CD))		W		L	W	
					W	L	W	
[[[AB]C]D]	((AB)(CD))	(((AB)C)D)				W	L	L

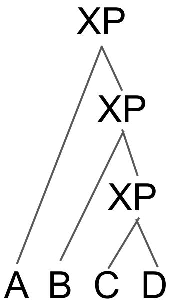
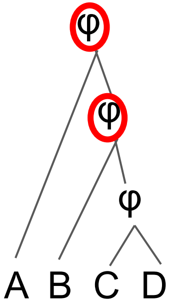
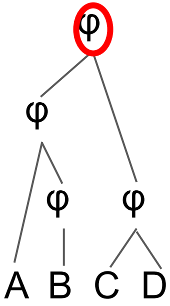
Match(XP,φ) prefers winner, but we already know **BinMax(φ,ω) >> Match(XP,φ)**

Support for L.12: *R-branching→isomorphic*

Input	**Winner**	**Loser**	ALIGNL (XP,φ)	**BINMIN (φ,ω)**	BINMAX (φ,br)	BINMAX (φ,ω)	MATCH (XP,φ)	ALIGNR (XP,φ)
[A[B[CD]]]	(A(B(CD)))	((AB)(CD))	W			L	W	
				W		L	W	
[A[[BC]D]]	(A((BC)D))	(A(BC)D)			W	L	W	
[[[AB]C]D]	((AB)(CD))	(((AB)C)D)				W	L	L

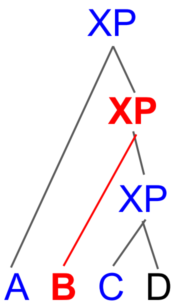
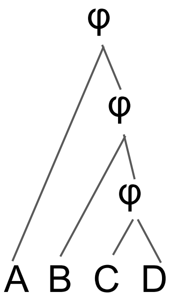
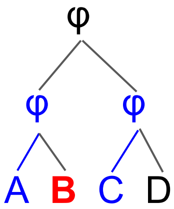
BINMIN(φ,ω) prefers winner; loser contains **unary ($_{\varphi}$B)**.

Support for L.12: *R-branching→isomorphic*

Input	**Winner**	**Loser**	ALIGNL (XP,φ)	BINMIN (φ,ω)	BINMAX (φ,br)	**BINMAX (φ,ω)**	MATCH (XP,φ)	ALIGNR (XP,φ)
[A[B[CD]]]	(A(B(CD)))	((AB)(CD))	W			L	W	
				W		L	W	
[A[[BC]D]]	(A((BC)D))	(A(BC)D)			W	L	W	
[[[AB]C]D]	((AB)(CD))	(((AB)C)D)				W	L	L

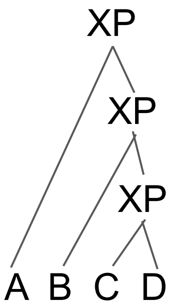
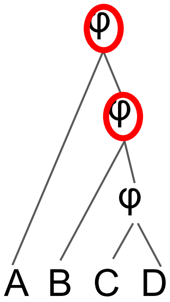
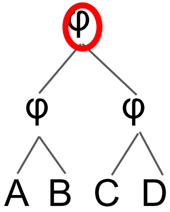
BINMAX(φ,ω) prefers loser.

Support for L.12: *R-branching→isomorphic*

Input	**Winner**	**Loser**	**AlignL (XP,φ)**	BinMin (φ,ω)	BinMax (φ,br)	BinMax (φ,ω)	Match (XP,φ)	AlignR (XP,φ)
			W			L	W	
[A[B[CD]]]	(A(B(CD)))	((A(B))(CD))		W		L	W	
[A[[BC]D]]	(A((BC)D))	(A(BC)D)			W	L	W	
[[[AB]C]D]	((AB)(CD))	(((AB)C)D)				W	L	L

AlignL(XP,φ) prefers winner; **[B↛(B** in loser.

Support for L.12: *R-branching→isomorphic*

Input	**Winner**	**Loser**	ALIGNL (XP,φ)	BINMIN (φ,ω)	BINMAX (φ,br)	**BINMAX (φ,ω)**	MATCH (XP,φ)	ALIGNR (XP,φ)
			W			L	W	
[A[B[CD]]]	(A(B(CD)))	((A(B))(CD))		W		L	W	
[A[[BC]D]]	(A((BC)D))	(A(BC)D)			W	L	W	
[[[AB]C]D]	((AB)(CD))	(((AB)C)D)				W	L	L

BINMAX(φ,ω) prefers loser.

S.Msp.Mps

Pure MATCH

Can we get the pattern without ALIGN?

Can we get the pattern without ALIGN?

- No!

Can we get the pattern without ALIGN?

- No!
- Consider a Pure MATCH system S.Msp.Mps

Can we get the pattern without ALIGN?

- No!
- Consider a Pure MATCH system S.Msp.Mps

(1) GEN.Msp.Mps
    = GEN.Msp.Asp

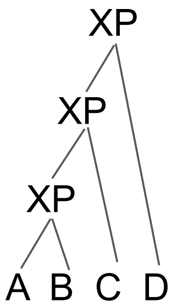
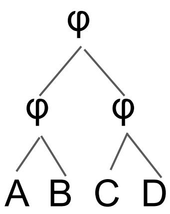
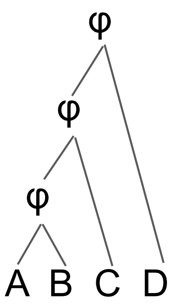
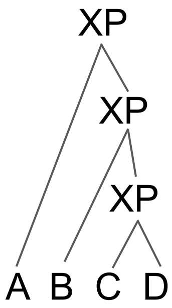
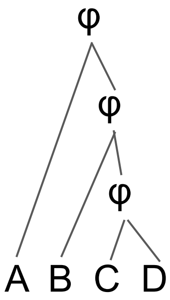
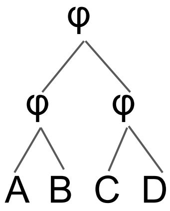
Can we get the pattern without ALIGN?

- No!
- Consider a Pure MATCH system S.Msp.Mps

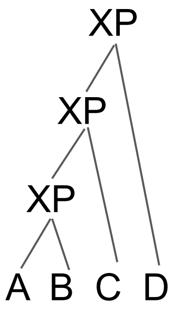
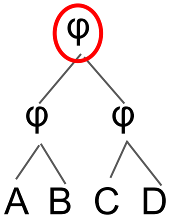
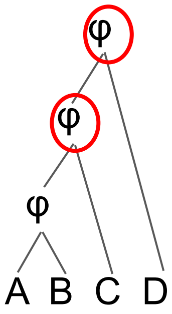
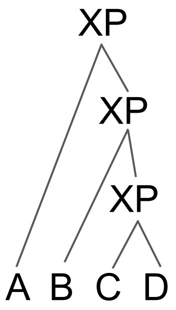
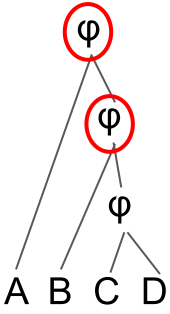
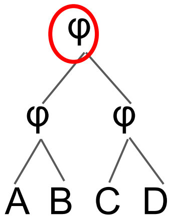
(1) GEN.Msp.Mps
 = GEN.Msp.Asp

(2) CON.Msp.Mps
 (a) Mapping constraints:
  (i) MATCH(XP,φ)
  (ii) MATCH(φ,XP): Assign one violation for every node of category φ in the prosodic tree such that there is no node of category XP in the syntactic tree that dominates all and only the same terminal nodes as φ.
 (b) Markedness constraints:
  (i) BINMIN(φ,ω)
  (ii) BINMAX(φ,ω)
  (iii) BINMAX(φ,branches)

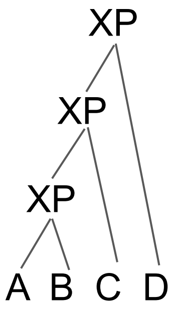
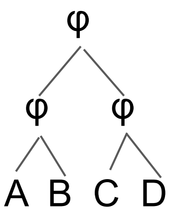
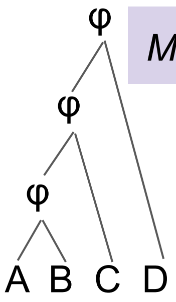
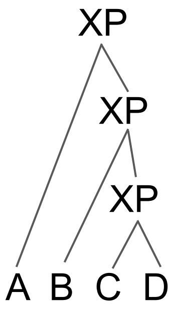
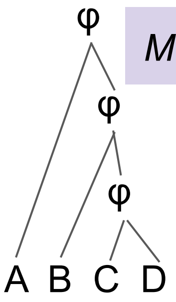
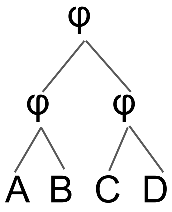
The Asymmetry Problem

Input	**Winner**	**Loser**	**Match (XP,φ)**	**Match (φ,XP)**	**BinMax (φ,ω)**	**BinMin (φ,ω)**	**BinMax (φ,br)**
			L	L	W	e	e
			W	W	L	e	e

The Asymmetry Problem

Input	**Winner**	**Loser**	MATCH (XP,φ)	MATCH (φ,XP)	**BINMAX (φ,ω)**	BINMIN (φ,ω)	BINMAX (φ,br)
			L	L	W	e	e
			W	W	L	e	e

The Asymmetry Problem

<table>
<tr><th>Input</th><th>Winner</th><th>Loser</th><th>MATCH (XP,φ)</th><th>MATCH (φ,XP)</th><th>BINMAX (φ,ω)</th><th>BINMIN (φ,ω)</th><th>BINMAX (φ,br)</th></tr>
<tr><td></td><td>Mismatch
</td><td>Match
</td><td>L</td><td>L</td><td>W</td><td>e</td><td>e</td></tr>
<tr><td></td><td>Match
</td><td>Mismatch
</td><td>W</td><td>W</td><td>L</td><td>e</td><td>e</td></tr>
</table>

The Asymmetry Problem

- To get the left/right asymmetry, we need at least one asymmetric constraint in CON.

The Asymmetry Problem

- To get the left/right asymmetry, we need at least one asymmetric constraint in CON.
- **MATCH** constraints are **symmetric**, and can't make the distinction.

The Asymmetry Problem

- To get the left/right asymmetry, we need at least one asymmetric constraint in CON.
- MATCH constraints are symmetric, and can't make the distinction.
- Asymmetries arise in Match Theory due to **markedness constraints.**

The Asymmetry Problem

- To get the left/right asymmetry, we need at least one asymmetric constraint in CON.
- MATCH constraints are symmetric, and can't make the distinction.
- Asymmetries arise in Match Theory due to markedness constraints**.**
- But all three of our **binarity** constraints are **symmetric**!

The Asymmetry Problem

- To get the left/right asymmetry, we need at least one asymmetric constraint in CON.
- MATCH constraints are symmetric, and can't make the distinction.
- Asymmetries arise in Match Theory due to markedness constraints.
- But all three of our binarity constraints are symmetric!
- Could we fix the problem with an asymmetric markedness constraint?

The Asymmetry Problem

- To get the left/right asymmetry, we need at least one asymmetric constraint in CON.
- MATCH constraints are symmetric, and can't make the distinction.
- Asymmetries arise in Match Theory due to markedness constraints.
- But all three of our binarity constraints are symmetric!
- Could we fix the problem with an asymmetric markedness constraint?
  - Perhaps, but not with one we're aware of.

The Asymmetry Problem

- To get the left/right asymmetry, we need at least one asymmetric constraint in CON.
- MATCH constraints are symmetric, and can't make the distinction.
- Asymmetries arise in Match Theory due to markedness constraints.
- But all three of our binarity constraints are symmetric!
- Could we fix the problem with an asymmetric markedness constraint?
  - Perhaps, but not with one we're aware of.
  - STRONGSTART doesn't work—in fact, it works against us here (see (24) in our paper).

The Asymmetry Problem

- To get the left/right asymmetry, we need at least one asymmetric constraint in CON.
- MATCH constraints are symmetric, and can't make the distinction.
- Asymmetries arise in Match Theory due to markedness constraints.
- But all three of our binarity constraints are symmetric!
- Could we fix the problem with an asymmetric markedness constraint?
  - Perhaps, but not with one we're aware of.
  - STRONGSTART doesn't work—in fact, it works against us here (see (24) in our paper).
  - Hypothetical STRONGEND doesn't work either (see (26) in our paper).

The Asymmetry Problem

- To get the left/right asymmetry, we need at least one asymmetric constraint in CON.
- MATCH constraints are symmetric, and can't make the distinction.
- Asymmetries arise in Match Theory due to markedness constraints**.**
- But all three of our binarity constraints are symmetric!
- Could we fix the problem with an asymmetric markedness constraint?
  - Perhaps, but not with one we're aware of.
  - STRONGSTART doesn't work—in fact, it works against us here (see (24) in our paper).
  - Hypothetical STRONGEND doesn't work either (see (26) in our paper).
- *Conclusion*: Unless we find a plausible asymmetric markedness constraint, **we need ALIGN constraints**.*

*Or some other, as yet undiscovered, asymmetric mapping constraint.

S.Asp.Aps

Pure ALIGN

Can we get the pattern without MATCH?

- No!
- Consider a Pure ALIGN system S.Asp.Aps

(1) GEN.Asp.Aps
    = GEN.Msp.Asp

Can we get the pattern without MATCH?

- No!
- Consider a Pure ALIGN system S.Asp.Aps

(1) GEN.Asp.Aps
= GEN.Msp.Asp

(2) CON.Asp.Aps
(a) Mapping constraints:

ALIGNL(XP,φ)	ALIGNL(φ,XP)
ALIGNR(XP,φ)	ALIGNR(φ,XP)

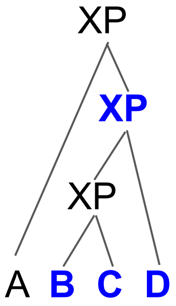
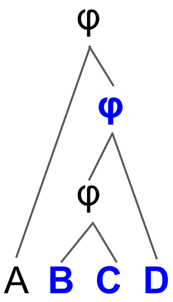
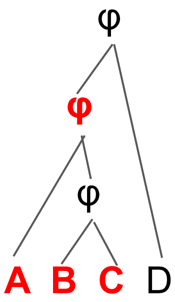
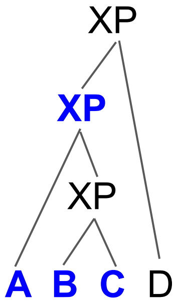
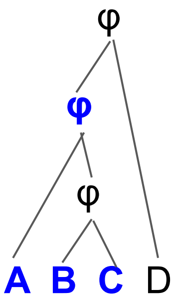
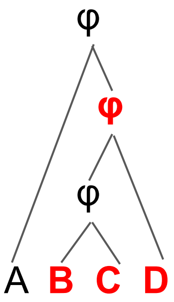
Can we get the pattern without MATCH?

- No!
- Consider a Pure ALIGN system S.Asp.Aps

(1) GEN.Asp.Aps
 = GEN.Msp.Asp

(2) CON.Asp.Aps
 (a) Mapping constraints:
 ALIGNL(XP,φ) ALIGNL(φ,XP)
 ALIGNR(XP,φ) ALIGNR(φ,XP)
 (b) Markedness constraints
 same binarity constraints as before

The Ambivalence Problem

Input	Winner	Loser	ALIGNL (XP,φ)	ALIGNR (XP,φ)	ALIGNL (φ,XP)	ALIGNR (φ,XP)	BINMAX (φ,ω)	BINMIN (φ,ω)	BINMAX (φ,br)
			e	e	e	e	e	e	e
			e	e	e	e	e	e	e

The Ambivalence Problem

- The Ambivalence Problem arises because ALIGN constraints can be satisfied without matching all XP's/φ's.

The Ambivalence Problem

- The Ambivalence Problem arises because ALIGN constraints can be satisfied without matching all XP's/φ's.
- For [A[[BC]D]], no need to match [BCD] to get perfect SP and PS alignment.

The Ambivalence Problem

- The Ambivalence Problem arises because ALIGN constraints can be satisfied without matching all XP's/φ's.
- For [A[[BC]D]], no need to match [BCD] to get perfect SP and PS alignment.
- **No markedness constraint**, standard or novel, could solve this problem; difference between [A[[BC]D]]→(A((BC)D)) and [[A[BC]]D]→((A(BC))D) comes down to **mapping (faithfulness)**.

The Ambivalence Problem

- The Ambivalence Problem arises because ALIGN constraints can be satisfied without matching all XP's/φ's.
- For [A[[BC]D]], no need to match [BCD] to get perfect SP and PS alignment.
- No markedness constraint, standard or novel, could solve this problem; difference between [A[[BC]D]]→(A((BC)D)) and [[A[BC]]D]→((A(BC))D) comes down to mapping (faithfulness).
- The mapping constraint **WRAP(XP) can't solve the problem**; all of the outputs in our systems satisfy it perfectly.

The Ambivalence Problem

- The Ambivalence Problem arises because ALIGN constraints can be satisfied without matching all XP's/φ's.
- For [A[[BC]D]], no need to match [BCD] to get perfect SP and PS alignment.
- No markedness constraint, standard or novel, could solve this problem; difference between [A[[BC]D]]→(A((BC)D)) and [[A[BC]]D]→((A(BC))D) comes down to mapping (faithfulness).
- The mapping constraint WRAP(XP) can't solve the problem; all of the outputs in our systems satisfy it perfectly.
- So we need a **MATCH** constraint.

Conclusion

Using SPOT, we have found that Japanese φ-phrasing involves **Match and Align**.

Direction for future research:

- Expanding past 4-word phrases, our systems make predictions for larger prosodic structures. Are these borne out?
- Are the other languages in the factorial typology of S.Msp.Asp empirically supported? (e.g. mirror-image Japanese?)