Effective Online Reordering with Arc-Eager Transitions

Ryosuke, Kohita. Hiroshi, Noji. Yuji, Matusmoto.

Nara Institute of Science and Technology

Overview

1. We focus on **transition-based parsing** and **online reordering**

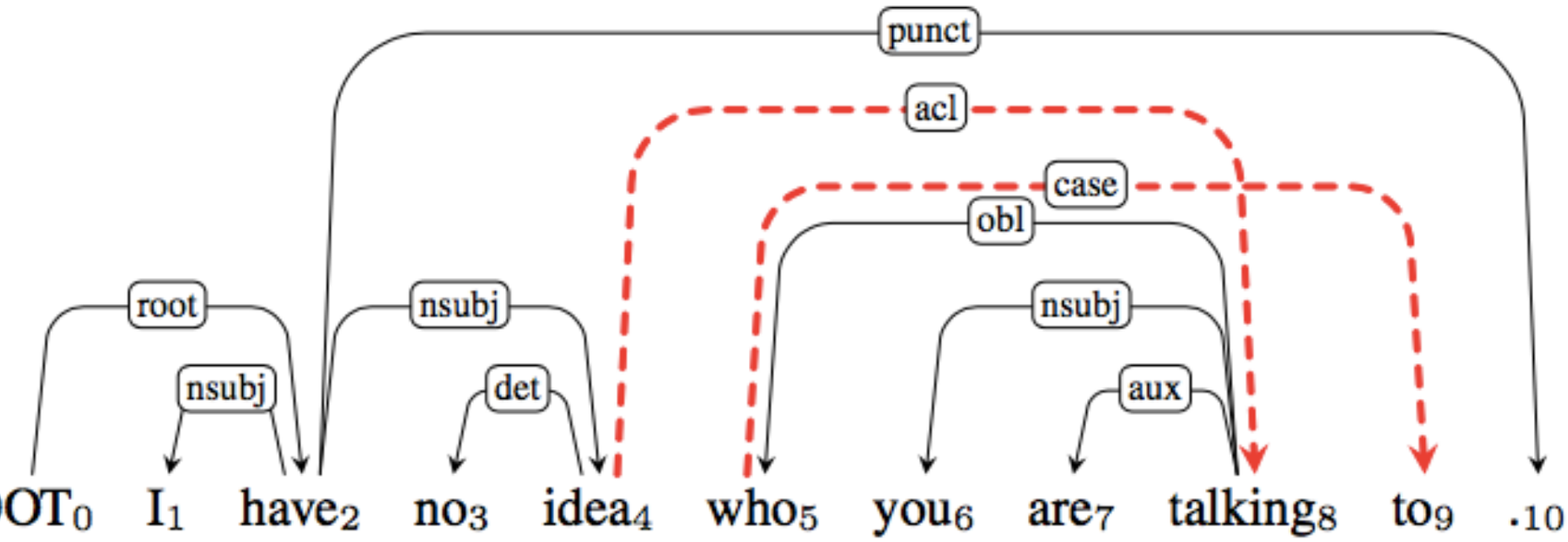
2. This is **a first attempt to incorporate SWAP operation into the Arc-Eager transition system**

3. We point out **the weakness of traditional SWAP transitions** and how our method challenges the problem

4. **Multilingual experiment with UD** shows the effectiveness of our proposed method

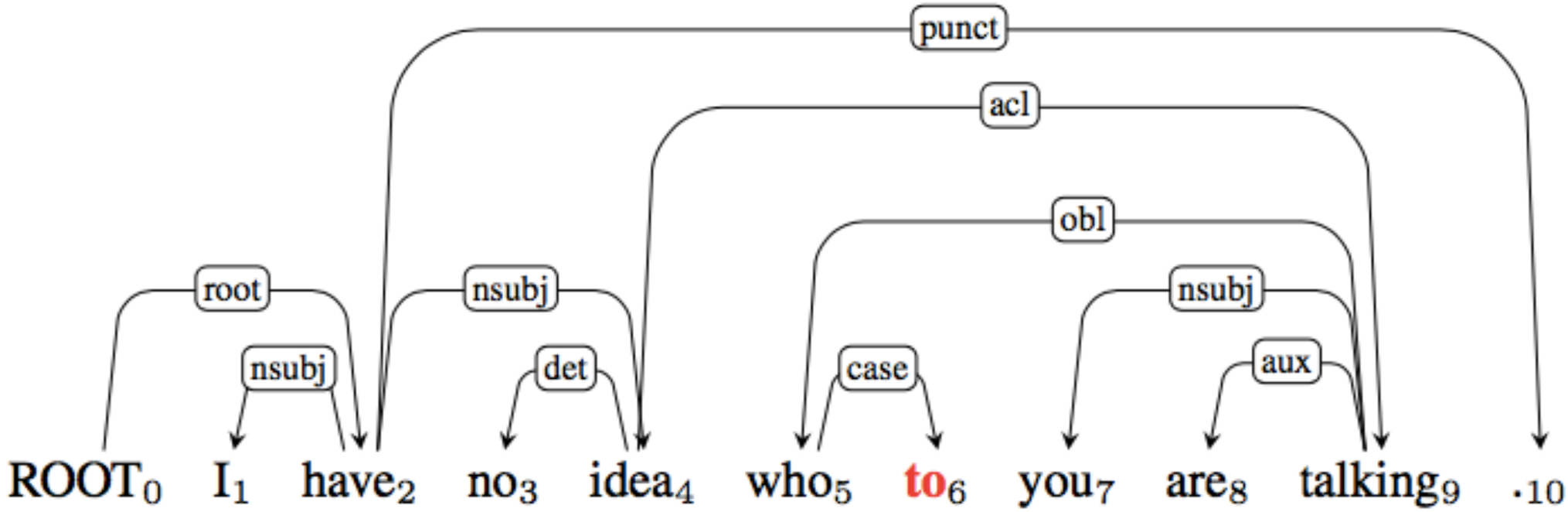
Introduction

Non-projectivity & Transition system



- Sentences with crossing arcs are called "***non-projective***"
- Transition-based parser basically can not process such sentences
- Its greedy left-to-right processing prohibits non-adjacent attachments

How does online reordering solve non-projective structures ?



- Any kinds of non-projective structures can be projectivised by reordering words appropriately (= ***projective order***)
- This reordering is processed **while parsing (= online)**

Why we focus on transition-based parsing and online reordering?

Comparing important algorithms

Algorithm	Incrementality	Unrestricted Non-projectives	Time Complexity	Standard Architect Single stack-buffer
Reordering [Nivre 09]+	**O**	**O**	O(n^2)	**O**
Attardi [Attardi 06]	**O**	X	O(n^2)	**O**
Covington [Covington 01]	**O**	**O**	O(n^2)	X
2-planar [Gomez 10]+	**O**	X	**O(n)**	X
Crossing-Interval [Pitler 15]+	**O**	X	**O(n)**	X
MST(graph) [McDonald 05]	X	**O**	O(n^2)	-

Incrementality

Algorithm	Incrementality
Reordering [Nivre 09]+	o
Attardi [Attardi 06]	o
Covington [Covington 01]	o
2-planar [Gomez 10]+	o
Crossing-Interval [Pitler 15]+	o
MST(graph) [McDonald 05]	x

- Transition-based algorithms can process a sentence incrementally
- Incremental processing is needed in some cases such as speech recognition

Cover ratio of non-projective structures

Algorithm	Incrementality	Unrestricted non-projective parsing
Reordering [Nivre 09]+	o	o
Attardi [Attardi 06]	o	x
Covington [Covington 01]	o	o
2-planar [Gomez 10]+	o	x
Crossing-Interval [Pitler 15]+	o	x
MST(graph) [McDonald 05]	x	o

- No-restriction is ideal
- Some algorithms sacrifice some types of non-projective structures for parsing efficacy

Time Complexity

Algorithm	Time Complexity	Standard Architect Single stack-buffer
Reordering [Nivre 09]+	O(n^2)	O
Attardi [Attardi 06]	O(n^2)	O
Covington [Covington 01]	O(n^2)	X
2-planar [Gomez 10]+	**O(n)**	X
Crossing-Interval [Pitler 15]+	**O(n)**	X
MST(graph) [McDonald 05]	O(n^2)	-

- Some algorithms work in linear-time thanks to their restrictions
- Other transition algorithms also work in near-linear time in most case
- MST always takes quadratic time

Standard Architecture : single stack and single buffer

Algorithm	Standard Architect Single stack-buffer
Reordering [Nivre 09]+	O
Attardi [Attardi 06]	O
Covington [Covington 01]	X
2-planar [Gomez 10]+	X
Crossing-Interval [Pitler 15]+	X
MST(graph) [McDonald 05]	-

- Standard architecture has been implemented in many systems
- Algorithms working on that are transparent with such existing systems

Online reordering seems very balanced

Algorithm	Incrementality	Unrestricted Non-projective Parsing	Time Complexity	Standard Architect Single stack-buffer
Reordering [Nivre 09]+	**o**	**o**	O(n^2)	**o**
Attardi [Attardi 06]	**o**	X	O(n^2)	**o**
Covington [Covington 01]	**o**	**o**	O(n^2)	X
2-planar [Gomez 10]+	**o**	X	**O(n)**	X
Crossing-Interval [Pitler 15]+	**o**	X	**O(n)**	X
MST(graph) [McDonald 05]	X	**o**	O(n^2)	-

Online reordering seems very balanced

Algorithm	Incrementality	Unrestricted Non-projective Parsing	Time Complexity	Standard Architect Single stack-buffer
Reordering [Nivre 09]+	**o**	**o**	O(n^2)	**o**

The advantages of online reordering

- Satisfying demands for **incremental settings**
- Covering **ALL** non-projective structures
- Running in **expected linear-time**
- **Simple** implementation and **transparent with existing systems**

Problem & Research question

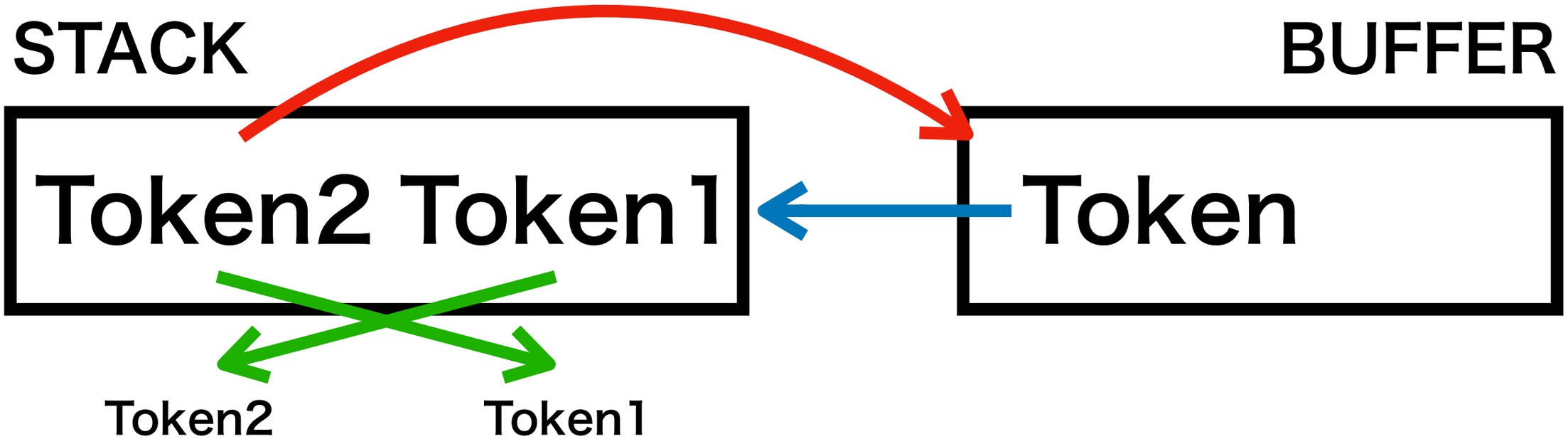
- Problem
  - Lack of further investigations about reordering ways
- Research question
  - Is the current strategy best way?
  - Is there more effective way of online reordering?

Traditional Approach:

Arc-**S**tandard **S**wap

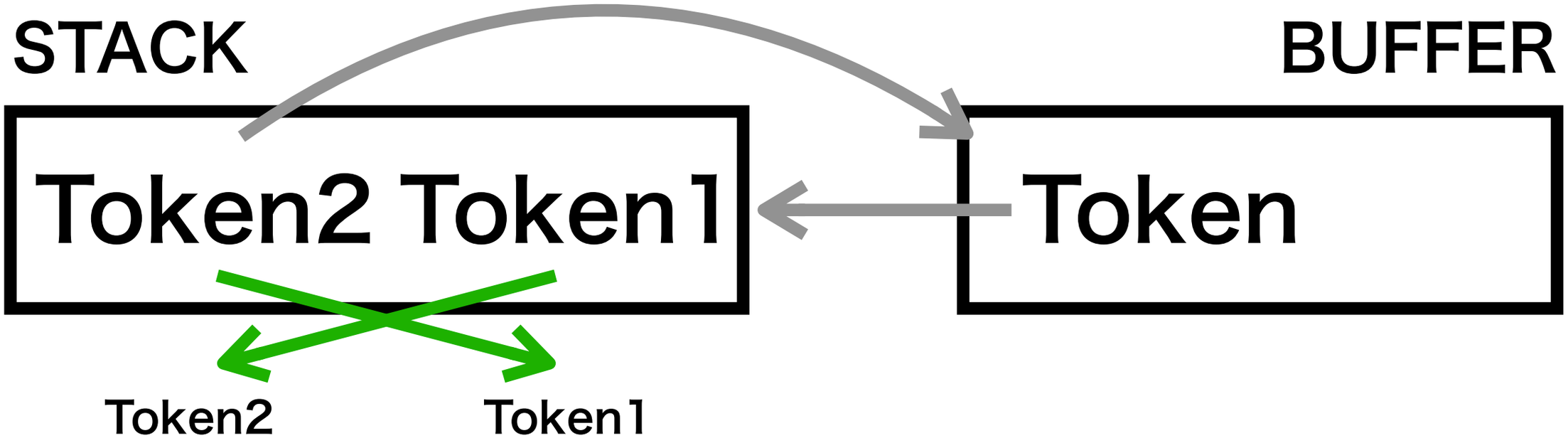
Proposed by Nivre, Kuhlmann & Hall 2009

Arc-**S**tandard **S**wap Transitions



- **RightArc**(**RA**), **LeftArc**(**LA**) : Connect top two tokens in the stack and reduce a dependent
- **Swap**(**SW**) : Push back a second token in the stack to the buffer
- **Shift**(**SH**) : Move a top token in the buffer to the stack

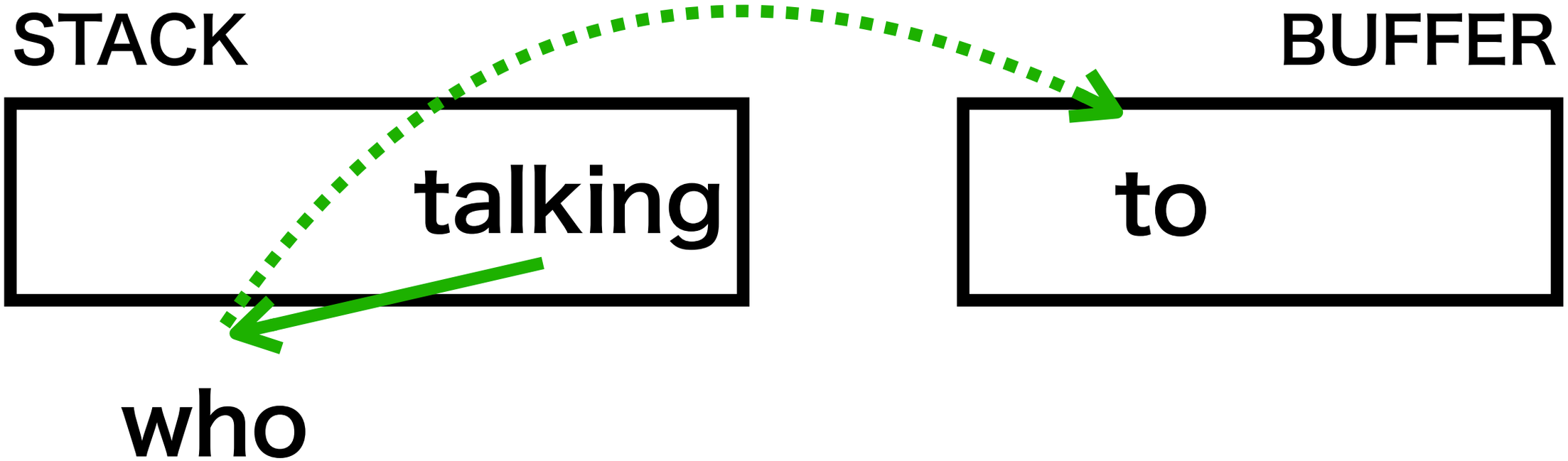
What we consider as trouble is …



- **RightArc**(**RA**), **LeftArc**(**LA**) : Connect top two tokens in the stack and **reduce a dependent**

Reduced tokens cannot have children anymore
If it has a dependent in the buffer … ?

Missing relationships between a reduced token and its dependents

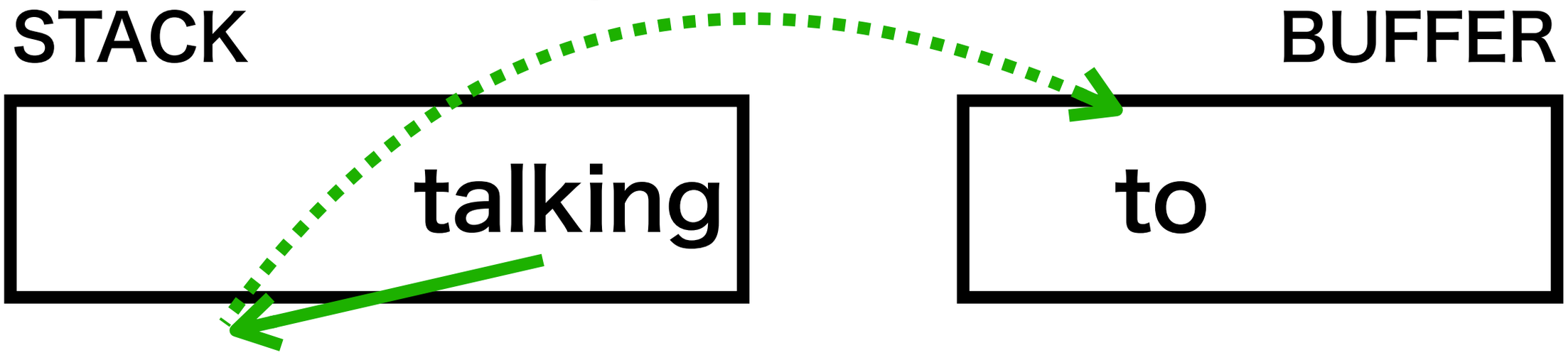


- Missing relationships is likely to occur in some cases …
  - Distant dependent (i.e. a dependent exists in buffer-depth)
  - Typical relationship (e.g. talking -> who : typical relative clause)

- **Parser cannot stop to attach such NEAR or LIKELY arcs, which causes missing far dependents**

For non-projective arcs,

the thing is more crucial



who

- ASS's bottom-up parsing will be difficult in some cases
  - Non-projective arcs tend to be **FAR** and **UNLIKELY**
  - The nature of ASS does not seem sound with non-projective structures
  - Distant dependent (far dependent exists in buffer's depth)
  - Typical relationship (e.g. talking -> who : typical relative clause)
- **Parser cannot stop to attach such NEAR or LIKELY arcs, which causes missing far dependents**

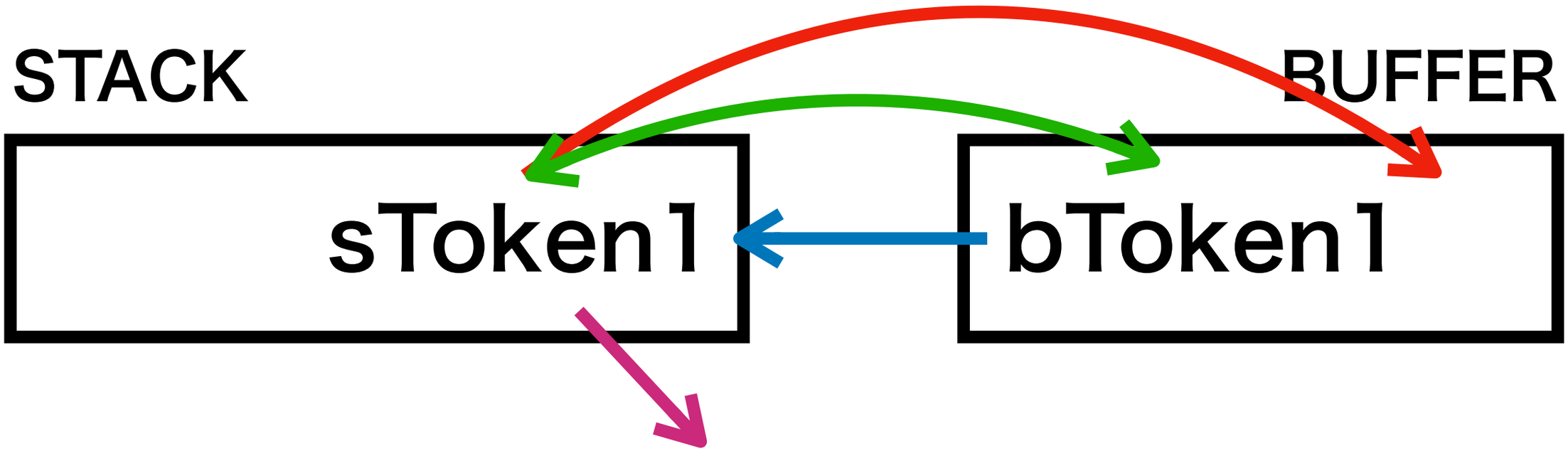
Summary of ASS's weakness

- **Simultaneous arc-attachment and reduce seems unmatched with non-projectivity**
  1. ASS needs to wait arc-attachments if a dependent still has children in the buffer
  2. Non-projective arcs are comparatively far and unlikely

Proposed Approach:

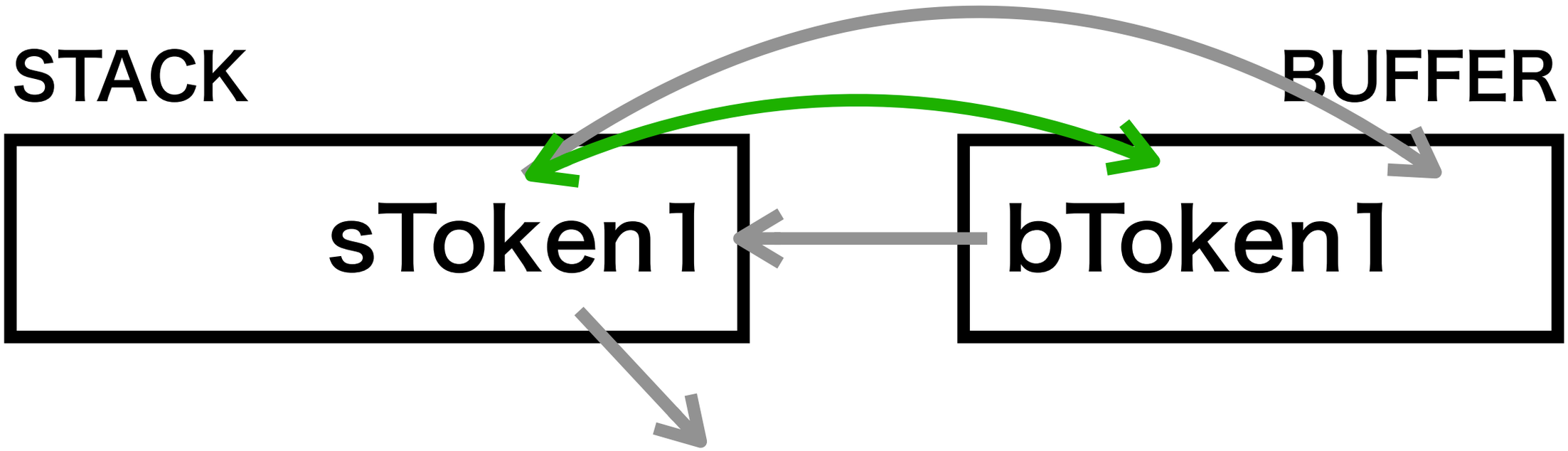
Stay-**E**ager **S**wap

Stay-**E**ager **S**wap Transitions



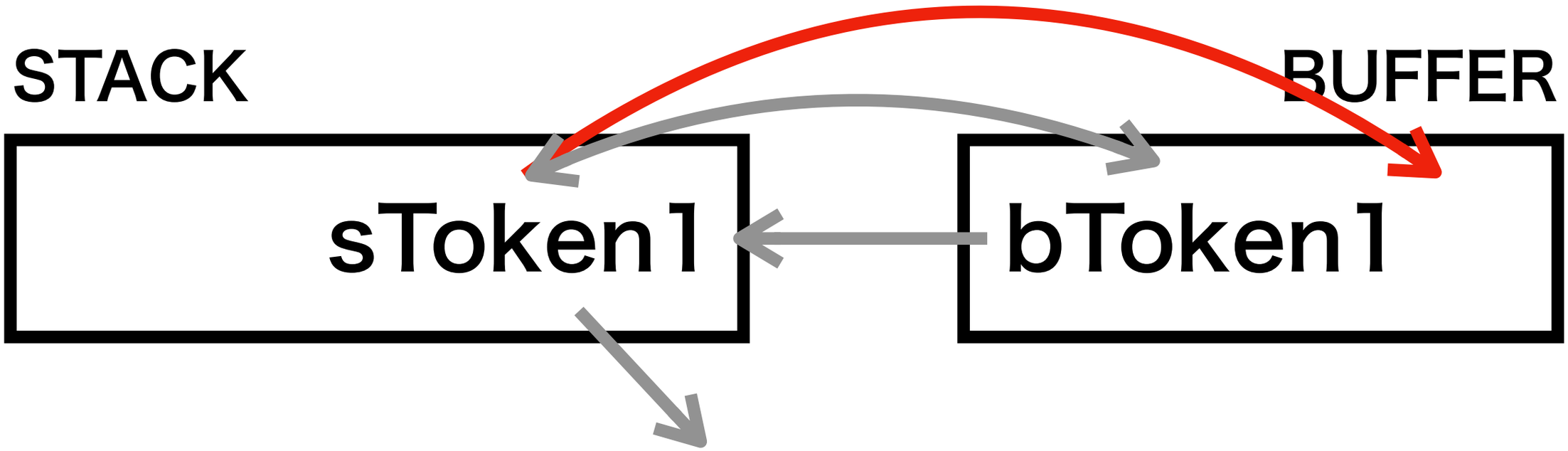
- StayRight(SR), StayLeft(SL) : Only connect the stack and buffer tops
- Swap(SW) : Push back stack top at the second position in buffer
- Shift(SH) : Move buffer top to stack
- Reduce(RD) : Pop stack top token

Stay-**E**ager **S**wap Transitions



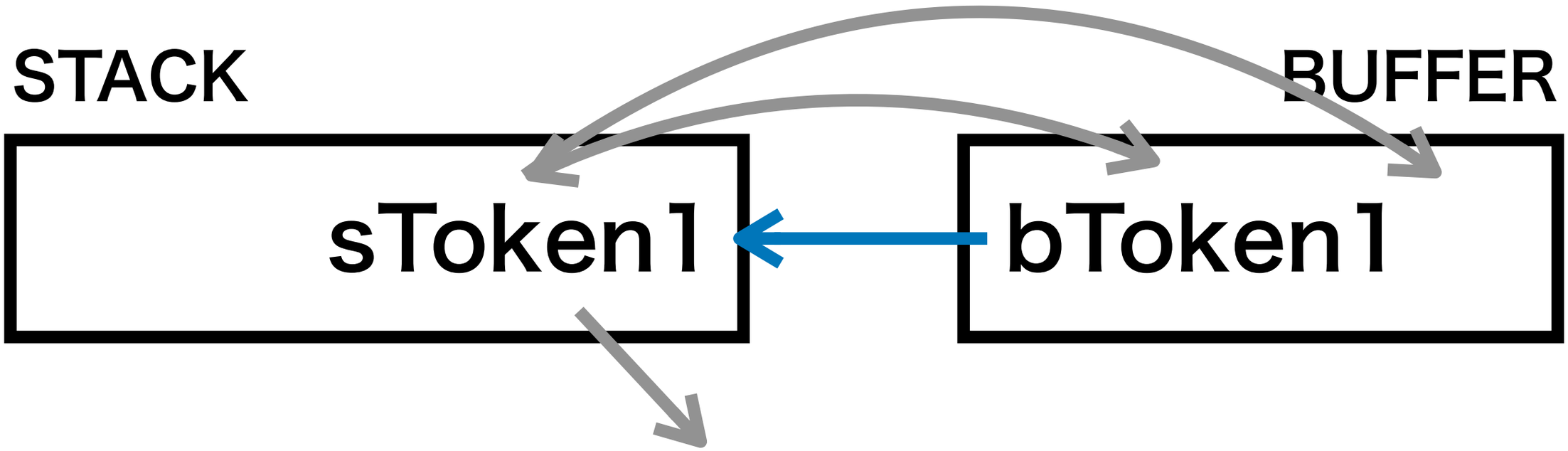
- **StayRight(SR), StayLeft(SL)** : **Only connect** the stack and buffer tops
- **Swap(SW)** : Push back stack top at the **second position in buffer**
- **Shift(SH)** : Move buffer top to stack
- **Reduce(RD)** : Pop stack top token

Stay-**E**ager **S**wap Transitions



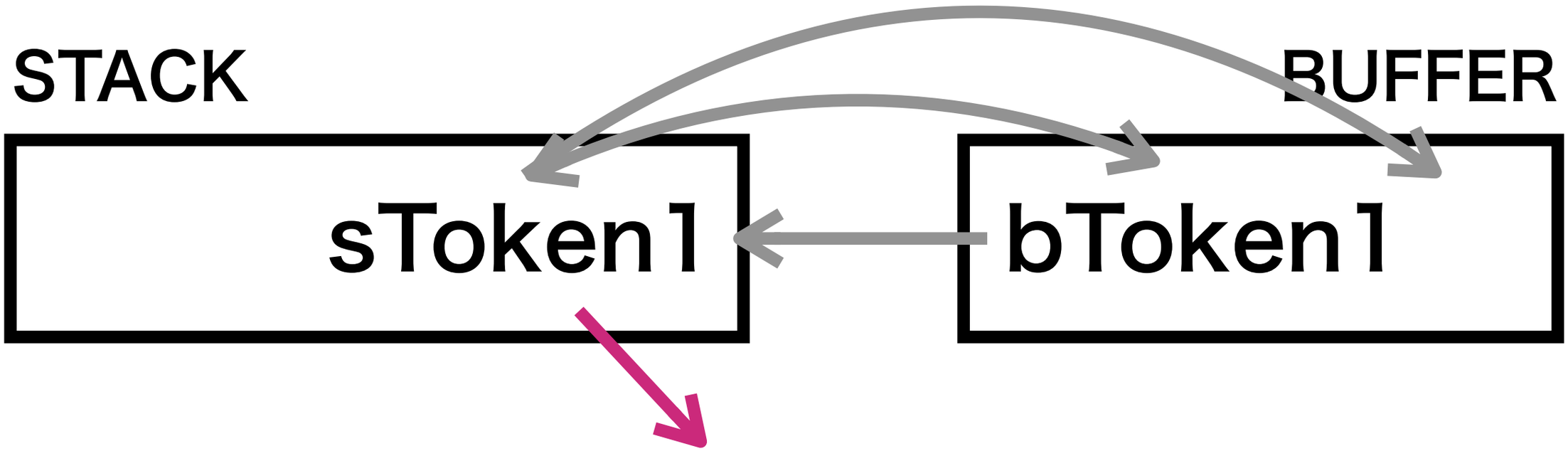
- **StayRight(SR), StayLeft(SL) : ONLY connect stack and buffer tops**
- **Swap(SW)** : Push back the stack top at the **second position** in the buffer
- **Shift(SH)** : Move buffer top to stack
- **Reduce(RD)** : Pop stack top token

Stay-**E**ager **S**wap Transitions



- **StayRight(SR), StayLeft(SL) : ONLY connect stack and buffer tops**
- **Swap(SW)** : Push back stack top at the **second position in buffer**
- **Shift(SH)** : Move the buffer top to the stack
- **Reduce(RD)** : Pop stack top token

Stay-**E**ager **S**wap Transitions



- **StayRight(SR), StayLeft(SL) : ONLY connect stack and buffer tops**
- **Swap(SW)** : Push back stack top at the **second position in buffer**
- **Shift(SH)** : Move buffer top to stack
- **Reduce(RD)** : Pop the stack top

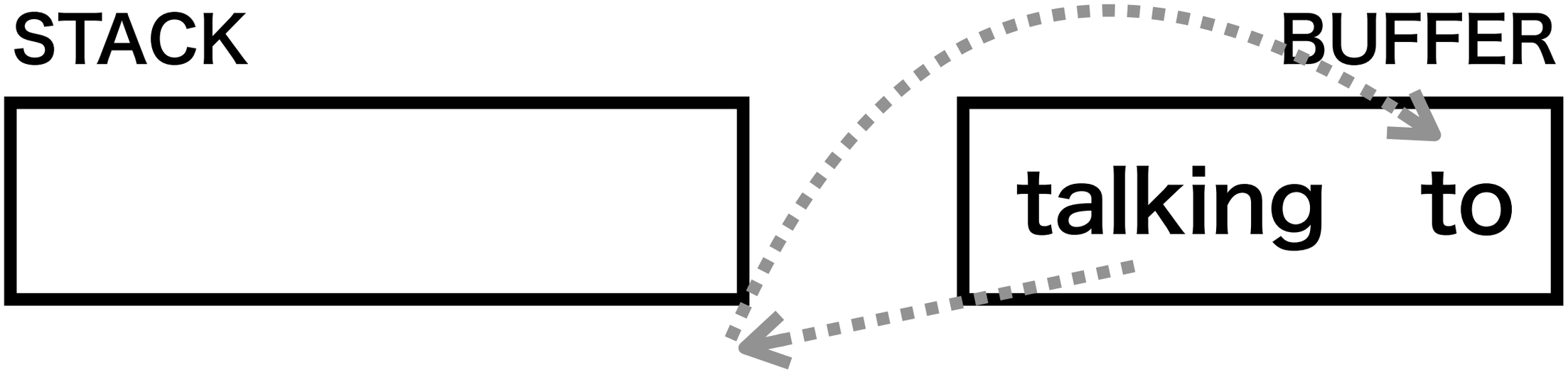
The modifications from ASS to SES

- **Reduce operation manages reduce timing**
  - Reduce is independent from arc-attachments
  - Reduce takes very important responsibility to avoid the problem of missing arcs like ASS

- **Arc-attachment can be applied greedy**
  - SES does not need to wait if it predicts a relationship between stack top and buffer top

Stay-Eager Swap with **non-static oracle**

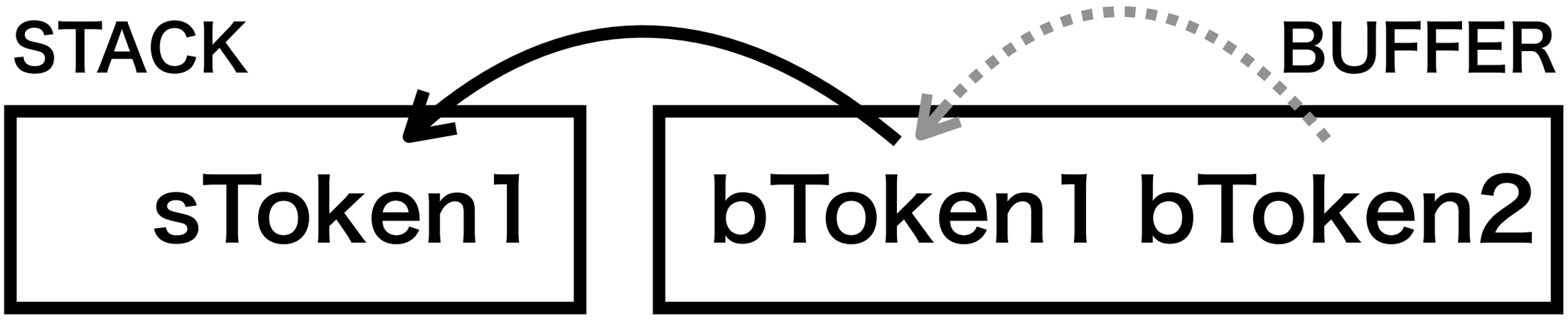
Static oracle



Who

- In static oracle, **RD is always applied as possible as early**
- Parser just learns a **new oracle sequence** causing missing arcs
- In the above case, the sequence “SL -> RD” represents a transition which causes missing dependents
- Parser needs to learn reduce in some good ways

Non-static oracle

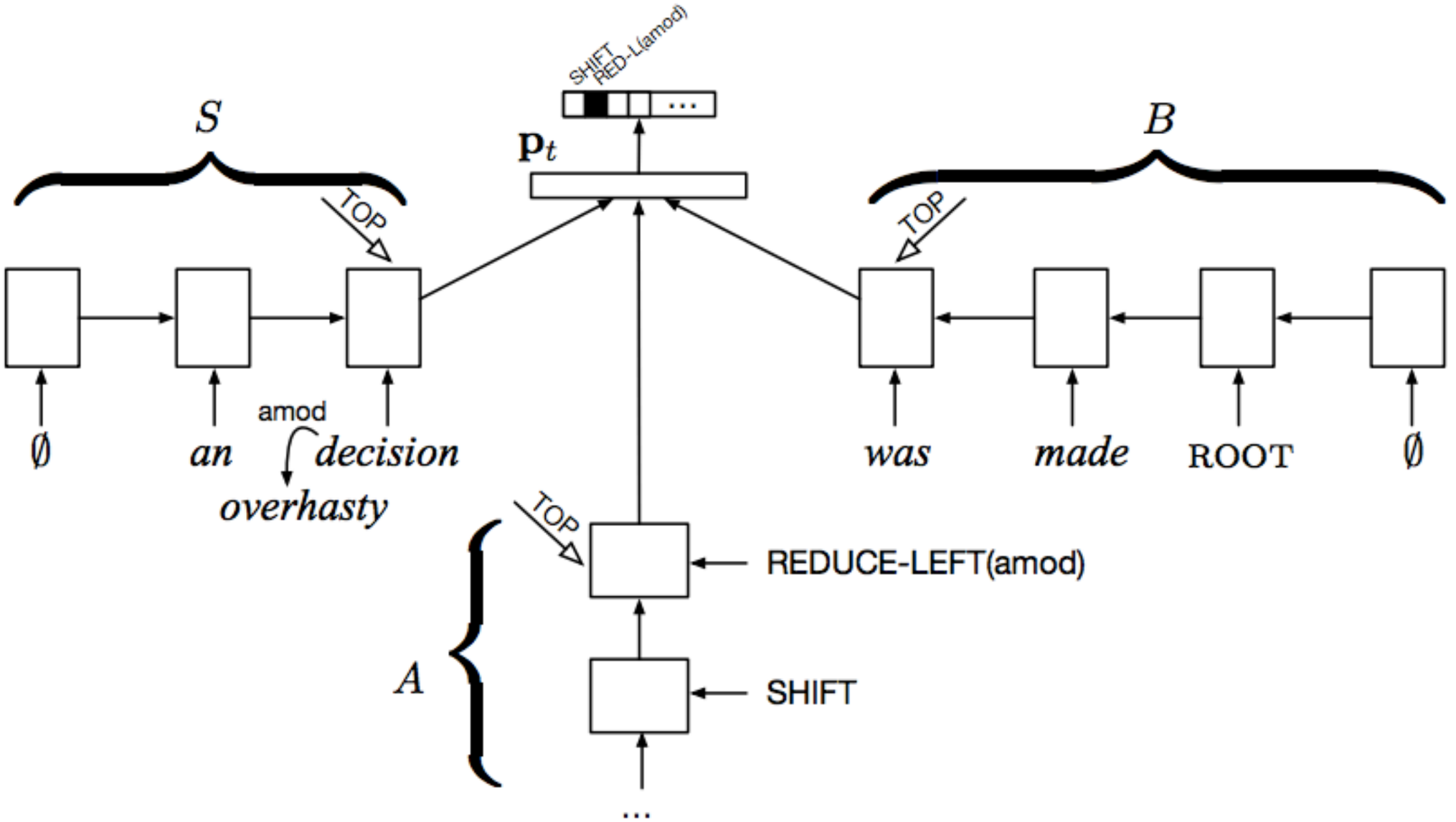


- When reduce can be postponed, a parser chooses shift stochastically
- **Motivation : Make a parser to look ahead**
- This is very simple and partial version of non-static oracle or dynamic oracle (e.g. Goldberg & Nivre 12)

Experiment

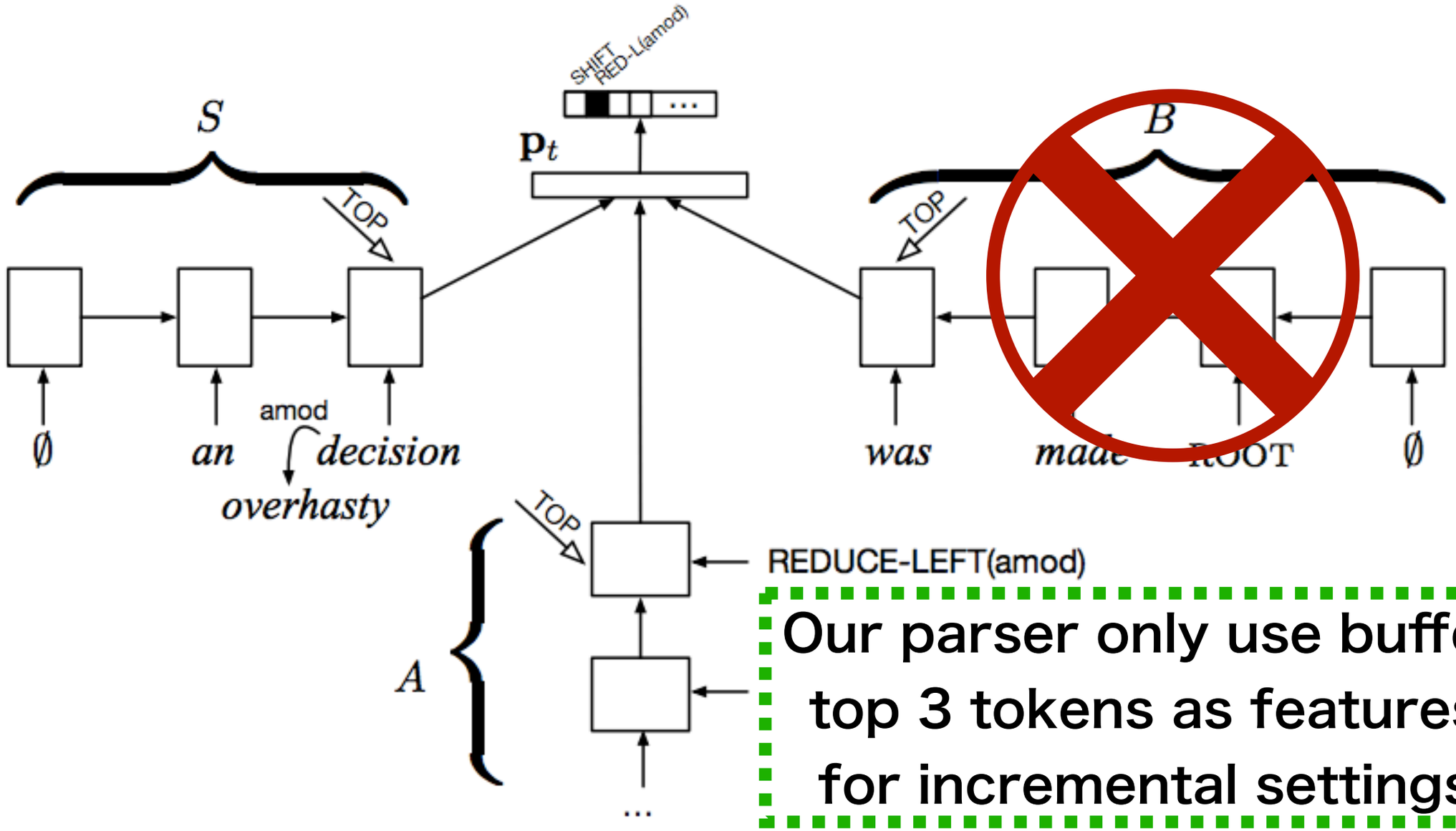
Settings : Parser

Stack-lstms parser proposed by Dyer et al., 2015



Settings : Parser

Stack-lstms parser proposed by Dyer et al., 2015



Our parser only use buffer top 3 tokens as features for incremental settings

Settings : Data

- **63 languages** in Universal Dependencies v2.0
  - Same set in pre-distributed languages in CoNLL UD shared task
- **Raw text analysis**
  - We use UDPipe v1.0 for sentence segment, word segment and morphological analysis in test session (Straka et.al, 15)

Settings : Procedure

- We compare three transitions
  - Arc-Standard Swap (**baseline**)
  - Static Stay-Eager Swap (**proposal**)
  - Non-static Stay-Eager Swap (**proposal**)
- We report Labeled Attachment Scores for …
  - All, only projective and only non-projective sentences
  - Results after grouping languages by their inclusion of non-projective sentences

Results

All languages & All sentences

ASS	static SES	non-static SES
67.59	67.56	**68.05**

All languages & **Non-projective** sentences

ASS	static SES	non-static SES
59.51	60.28	**61.05**

Language groups & All sentences

Inclusion of non-projectivity	ASS	non-static SES
HIGH (20% ~)	62.07	**62.96**
MID (10 ~ 20%)	67.27	**67.64**
LOW (5 ~ 10%)	73.23	**73.73**
VERYLOW (~ 5%)	67.49	**67.65**

Language groups & Projective sentences

Inclusion of non-projectivity	ASS	non-static SES
HIGH (20% ~)	66.38	**66.85**
MID (10 ~ 20%)	68.35	**68.59**
LOW (5 ~ 10%)	74.17	**74.57**
VERYLOW (~ 5%)	67.91	**68.01**

Language groups & Non-projective sentences

Inclusion of non-projectivity	ASS	non-static SES
HIGH (20% ~)	57.19	**58.60**
MID (10 ~ 20%)	62.04	**63.02**
LOW (5 ~ 10%)	66.21	**67.66**
VERYLOW (~ 5%)	59.74	**62.56**

Discussion

Remaining Works

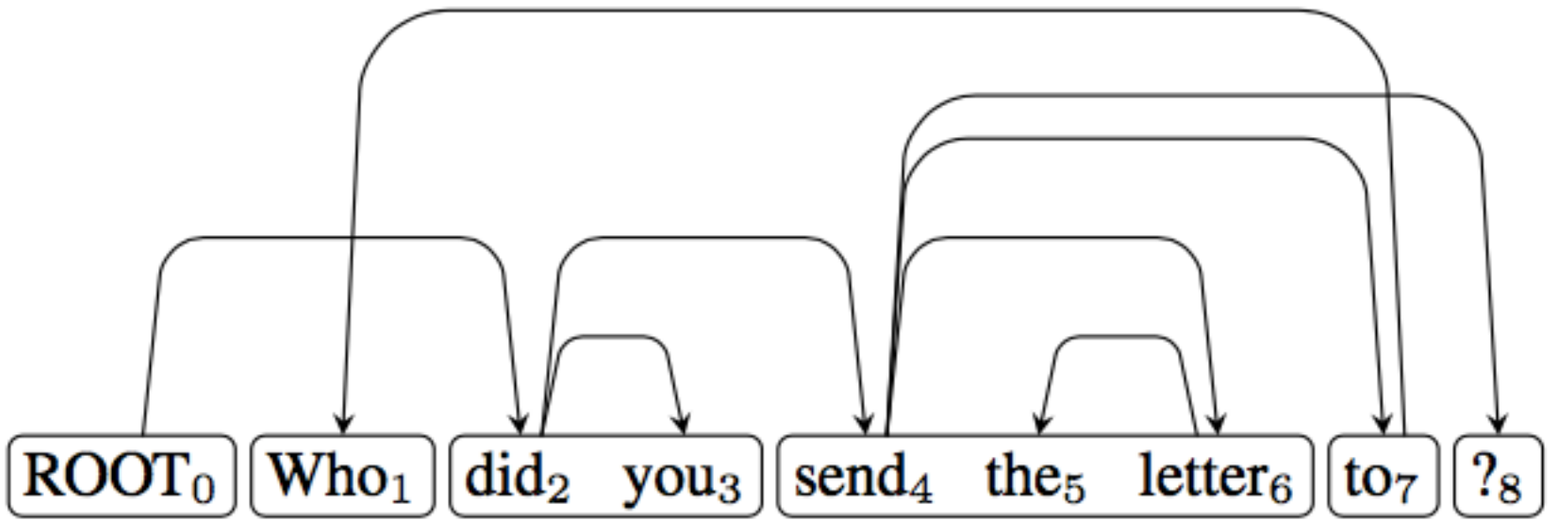
1. Why Arc-Eager? Stay-arc transition can be also combined with Arc-Standard ?

2. How about number of swap in SES compared to ASS ?

Why Arc-Eager ?

- To keep reduce operation itself simple
  - Arc-Eager reduces **only the stack top**
    - Arc-Standard reduces **the stack top and second, which requires two types of reduce operation**
- We wanted to avoid such complexity
- The incorporation is applicable, one of the future works

How’s number of swap?



- SES takes slightly large number of swap than ASS
- ASS can reduce swap by building local subtrees first, which is available because of its nature of bottom-up parsing
- It is not permitted for SES because it attaches right-branches in top-down manner, only left-branches are connected in bottom-up manner

Conclusion

A. We focus on some properties of online reordering

- 1) Incrementality
- 2) Covering all non-projective structures
- 3) Time complexity
- 4) Working on the standard transition architecture

B. Our proposed method got better results for projective and non-projective sentences in raw text analysis

C. Future works are …

- 1) Incorporation of stay-arc and ASS
- 2) Reducing number of swap in SES
- 3) Formal description to compare ASS and SES

Appendix

Oracle

Transition	Configuration	Condition
STAY-LEFT$_l$	$(\sigma \vert i, j \vert \beta, A)$	$(j, l, i) \in A_g$
STAY-RIGHT$_l$	$(\sigma \vert i, j \vert \beta, A)$	$(i, l, j) \in A_g$
SWAP	$(\sigma \vert i, j \vert \beta, A)$	$\mathit{isCross}(i, j)$
REDUCE	$(\sigma \vert i, \ \beta, A)$	$(\cdot, \cdot, i) \in A \wedge \forall h. \forall j. ((i, h, j) \in A_g \rightarrow (i, h, j) \in A)$
SHIFT	$(\sigma, \ j \vert \beta, A)$	$\forall i. \forall l. ((j, l, i) \in A_g \wedge j < i) \rightarrow (j, l, i) \in A$

[1] $\mathit{isCross}(i, j)$ returns true if i and j are two endpoints of two crossing arcs yet unattached. Formally: $(\exists b. b \in \beta \wedge ((b, \cdot, i) \in A_g \wedge (b, \cdot, i) \notin A \vee (i, \cdot, b) \in A_g \wedge (i, \cdot, b) \notin A)) \wedge (\exists s. s \in \sigma \wedge ((s, \cdot, j) \in A_g \wedge (s, \cdot, j) \notin A \vee (j, \cdot, s) \in A_g \wedge (j, \cdot, s) \notin A))$.

Right-Root (aux function 1)

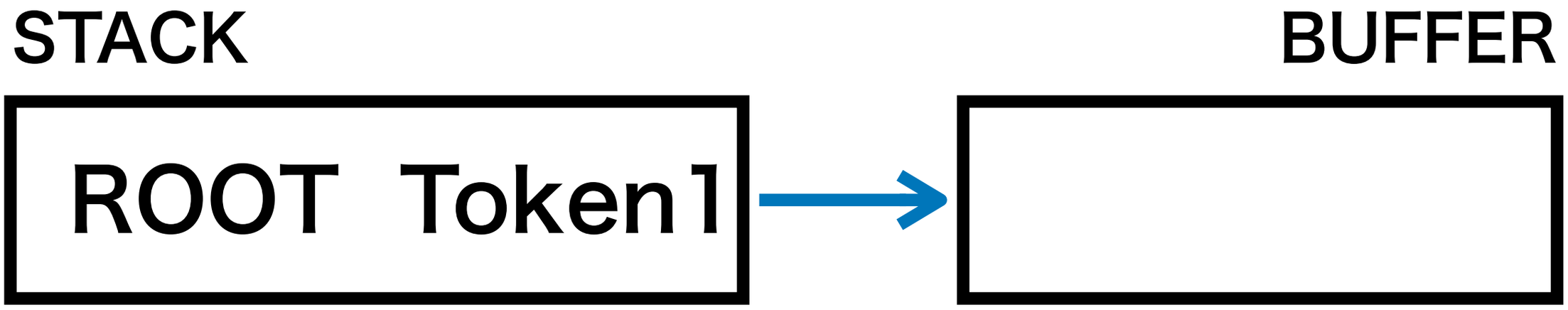
- Arc-Eager transitions decide sentence-root just after finished left side of a tree

- It sometimes decide sentence-root too early in some case such as verbal noun subject

  - e.g. *Parsing sentences is running on computers.*

- Right-Root action permit it to delay sentence-root decision, which is motivated to make a parser to look ahead

Unshift (aux function 2)



- Arc-Eager does not guarantee to output a tree
- Parser will fall into the configuration where stack top token has no head but buffer is empty
- Escape from such configuration by pushing back stack top token to buffer

LAS of HIGH languages

Language	Non-proj ratio	All sentences			
		ASS	SES		UDpipe
			static	non-static	
grc	64.40%	49.18 (51.28)	50.00 (53.04)	**50.10** (**53.85**)	56.04
la	47.50%	37.78 (37.82)	38.32 (42.50)	**41.22** (**43.27**)	43.77
grc_proiel	37.92%	60.67 (63.74)	**60.92** (**64.69**)	60.66 (63.77)	65.22
la_ittb	35.87%	72.29 (75.20)	71.98 (75.01)	**72.61** (**75.64**)	76.98
eu	31.80%	**66.61** (67.67)	65.68 (67.66)	65.74 (**68.96**)	69.15
en_lines	29.54%	70.37 (72.16)	71.71 (72.59)	**71.98** (**73.02**)	72.94
la_proiel	28.30%	50.56 (54.38)	53.71 (54.72)	**53.87** (**56.18**)	57.54
nl_lassysmall	28.13%	76.48 (76.25)	77.35 (77.78)	**77.63** (**77.83**)	78.15
pt	23.69%	**77.12** (**78.01**)	74.89 (75.49)	75.07 (77.75)	82.11
de	23.23%	65.11 (**66.94**)	63.35 (66.64)	**65.13** (66.83)	69.11
gl_treegal	23.00%	62.51 (63.78)	64.09 (63.66)	**64.55** (**64.48**)	65.82
nl	22.92%	**64.56** (65.69)	62.87 (66.86)	64.26 (**67.99**)	68.90
got	22.67%	54.07 (57.40)	55.46 (57.71)	**56.66** (**58.25**)	59.81
hu	21.60%	61.64 (62.72)	60.61 (62.44)	**61.94** (**63.45**)	64.30

LAS of MID languages

cu	18.93%	58.50 (60.41)	**61.17** (62.77)	60.66 (**63.62**)	62.76
ur	18.88%	**75.91** (**76.70**)	75.46 (76.46)	75.85 (76.49)	76.69
sv_lines	18.38%	71.48 (72.77)	71.18 (72.58)	**72.14** (**73.95**)	74.29
et	16.87%	56.02 (56.26)	**56.33** (56.57)	56.10 (**57.64**)	58.79
da	16.46%	70.23 (72.03)	69.63 (70.24)	**70.38** (**72.31**)	73.38
sl	15.99%	78.58 (79.85)	**78.72** (79.49)	**78.72** (**79.95**)	81.15
cs_cltt	15.76%	68.78 (68.66)	69.97 (70.06)	**70.52** (**71.30**)	71.64
cs_cac	12.74%	79.92 (**81.56**)	**80.10** (81.20)	79.42 (81.18)	82.46
hi	12.53%	**85.29** (**85.82**)	84.22 (85.56)	84.38 (85.79)	86.77
sl_sst	12.18%	41.45 (42.67)	**42.40** (44.15)	41.69 (**44.40**)	46.45
tr	12.10%	**52.67** (51.56)	52.00 (52.54)	**52.78** (52.49)	53.19
cs	11.98%	77.82 (**80.43**)	**78.13** (79.62)	78.01 (80.36)	82.87
el	11.62%	76.89 (78.31)	76.67 (77.72)	**77.73** (**78.67**)	79.26
ar	11.62%	63.79 (64.80)	**64.97** (**65.04**)	64.27 (64.74)	65.30
kk	11.56%	**21.83** (21.75)	18.43 (19.59)	20.08 (**22.84**)	24.51
fr	11.54%	**78.05** (**79.52**)	77.70 (78.86)	77.97 (79.43)	80.75
ca	10.94%	82.94 (83.86)	83.28 (84.02)	**83.40** (**84.33**)	85.39
ga	10.13%	58.58 (60.29)	60.50 (61.30)	**61.34** (**62.03**)	61.52
es	10.09%	79.48 (80.42)	79.64 (**80.81**)	**79.71** (80.49)	81.47

LAS of LOW languages

ro	9.74%	76.66 (78.25)	77.51 (78.34)	**77.75** (**78.92**)	79.88
es_ancora	9.65%	81.43 (83.18)	**81.99** (82.88)	81.72 (**83.21**)	83.78
ko	9.61%	**73.70** (**74.24**)	72.11 (72.80)	72.33 (73.39)	59.09
ru	9.48%	71.08 (73.84)	**72.49** (73.33)	72.17 (**74.85**)	74.03
ru_syntagrus	9.21%	83.89 (85.33)	**84.26** (84.95)	84.17 (**85.42**)	86.76
no_nynorsk	9.20%	**78.69** (79.52)	78.26 (78.78)	78.13 (**79.56**)	81.56
hr	9.17%	**73.74** (**76.08**)	73.47 (75.63)	73.61 (75.41)	77.18
fr_sequoia	8.77%	77.22 (78.69)	77.09 (77.61)	**78.19** (**79.16**)	79.98
uk	7.95%	57.85 (58.67)	58.97 (59.84)	**59.80** (**61.23**)	60.76
no_bokmaal	7.63%	79.81 (80.69)	79.75 (80.37)	**80.36** (**81.31**)	83.27
fa	7.17%	74.83 (**77.89**)	74.61 (77.42)	**75.94** (77.87)	79.24
fi_ftb	7.02%	70.95 (**73.15**)	71.07 (72.03)	**71.30** (72.75)	74.03
lv	6.64%	55.54 (57.10)	54.50 (56.76)	**56.47** (**57.35**)	59.95
fi	6.24%	**71.55** (72.37)	70.45 (**72.62**)	71.53 (72.18)	73.75
id	5.75%	71.48 (73.78)	72.17 (**74.16**)	**72.49** (74.04)	74.61

LAS of VERYLOW languages

it	4.98%	83.06 (84.68)	83.61 (84.30)	**83.74** (**84.92**)	85.28
pt_br	4.90%	**83.59** (**84.39**)	83.38 (83.92)	83.35 (84.04)	85.36
ug	4.78%	31.54 (34.02)	33.09 (33.04)	**34.40** (**34.57**)	34.18
sk	4.34%	70.13 (70.76)	**70.76** (**70.99**)	70.71 (70.69)	72.75
fr_partut	4.25%	75.86 (76.47)	76.81 (77.10)	**77.43** (**77.97**)	77.38
en_partut	4.00%	71.00 (72.61)	71.04 (72.43)	**71.78** (**73.44**)	73.64
en	3.71%	**72.53** (73.56)	72.51 (73.81)	72.29 (**73.90**)	75.84
vi	3.25%	36.37 (36.37)	36.11 (37.00)	**36.68** (**37.47**)	37.47
bg	2.87%	**81.24** (**82.02**)	79.92 (81.32)	80.58 (81.79)	83.64
sv	2.30%	74.09 (74.67)	73.65 (74.29)	**74.24** (**75.35**)	76.73
he	1.83%	55.02 (56.58)	54.91 (56.21)	**56.33** (**56.90**)	57.23
zh	1.40%	**53.55** (**55.23**)	51.65 (53.94)	51.93 (53.11)	57.40
pl	0.27%	76.80 (78.02)	**77.70** (**78.09**)	77.26 (77.75)	78.78
gl	0.00%	76.16 (77.43)	**77.24** (77.68)	77.04 (**78.35**)	77.31
ja	0.00%	**71.47** (**72.38**)	63.91 (68.59)	66.96 (71.77)	72.21

UDPipe got much higher score

Sentence type	Beam search	ASS	static SES	non-static SES	UDPipe
ALL	X	67.59	67.56	**68.05**	70.34

All languages & All Sentences

Sentence type	Beam search	ASS	static SES	non-static SES
ALL	X	67.59	67.56	**68.05**
	O	68.93	68.88	**69.55**

All languages & Non-projective sentences

Sentence type	Beam search	ASS	static SES	non-static SES
Only non-projective	X	59.51	60.28	**61.05**
	O	61.18	61.98	**62.36**