Incremental Discontinuous Phrase Structure Parsing with the GAP Transition

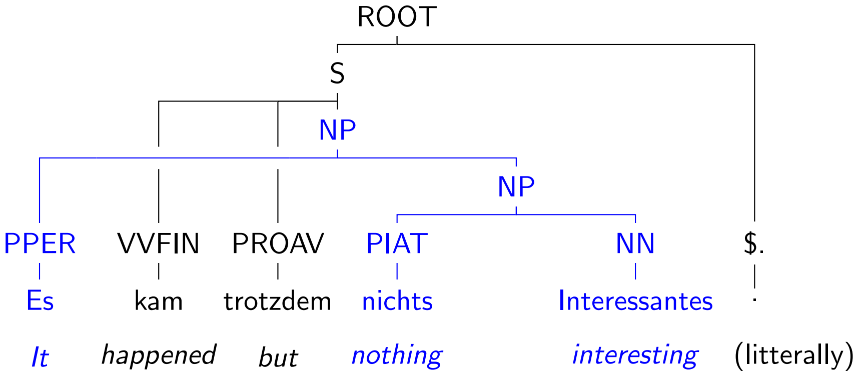
Maximin Coavoux[1,2] – Benoît Crabbé[1,2,3]

[1]Univ Paris Diderot – Sorbonne Paris Cité (SPC)
[2]Laboratoire de Linguistique Formelle (LLF, CNRS)
[3]Institut Universitaire de France (IUF)

EACL – Valencia – April 2017

Discontinuous Constituency Trees: Example

- Task: predict (potentially) discontinuous constituency trees



- Translation: *But nothing interesting happened* (Negra Corpus)

Discontinuous Constituency Trees: What for?

Wide range of phenomena

- Long distance extractions: some relative clauses, questions
- Dislocations
- Cross serial dependencies

Syntactical discontinuities are rather **frequent**

- languages with some degree of word order flexibility: 30% of sentences in German treebanks (Maier and Lichte, 2009)
- configurational languages: 20% of sentences in Discontinuous Penn Treebank (Evang and Kallmeyer, 2011)

Annotation strategies:

- Use **empty categories** (traces), coindexation (PennTB)
- Use **crossing branches** (Negra, Tiger) $\rightarrow$ Discontinuous trees

Discontinuous Parsing

Standard constituency parsing focused on projective trees

- remove traces/empty categories: too hard
- projectivize trees

Discontinuous Parsing

Standard constituency parsing focused on projective trees

- remove traces/empty categories: too hard
- projectivize trees

Approaches to discontinuous parsing

- Probabilistic grammar, CKY-like decoding
  - exact parsing has high polynomial complexity $\mathcal{O}(n^{3f})$
  - does not scale to full corpora
  - limited accuracy

Discontinuous Parsing

Standard constituency parsing focused on projective trees

- remove traces/empty categories: too hard
- projectivize trees

Approaches to discontinuous parsing

- Probabilistic grammar, CKY-like decoding
  - exact parsing has high polynomial complexity $\mathcal{O}(n^{3f})$
  - does not scale to full corpora
  - limited accuracy
- Transition based methods
  - Easy first (Versley, 2014), Swap action (Maier, 2015)
  - faster, scalable

Discontinuous Parsing

Standard constituency parsing focused on projective trees

- remove traces/empty categories: too hard
- projectivize trees

Approaches to discontinuous parsing

- Probabilistic grammar, CKY-like decoding
  - exact parsing has high polynomial complexity $\mathcal{O}(n^{3f})$
  - does not scale to full corpora
  - limited accuracy
- Transition based methods
  - Easy first (Versley, 2014), Swap action (Maier, 2015)
  - faster, scalable
- Reduction to dependency parsing
  - Fernández-González and Martins (2015), Hall and Nivre (2008)
  - tree conversion from const to dep
  - most successful approach so far

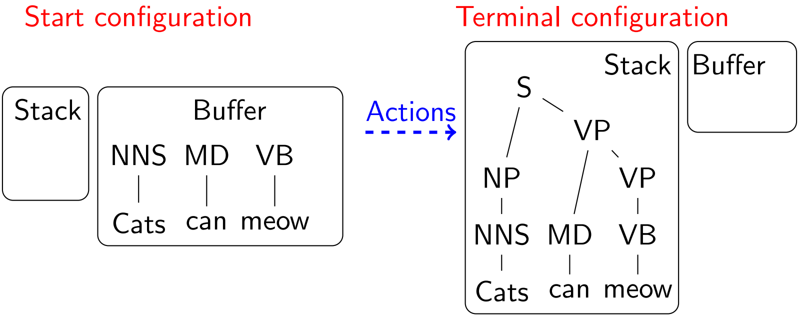
Contributions

- New transition system for discontinuous constituency parsing **Shift-Reduce+Gap**
  - Approximate parsing (drop grammaticality constraints)
  - Efficient (linear time parsing)
  - State-of-the-art results on 2 German treebanks with a perceptron
- Empirical comparison with previous best transition system (Shift-Reduce+Swap; Maier, 2015)

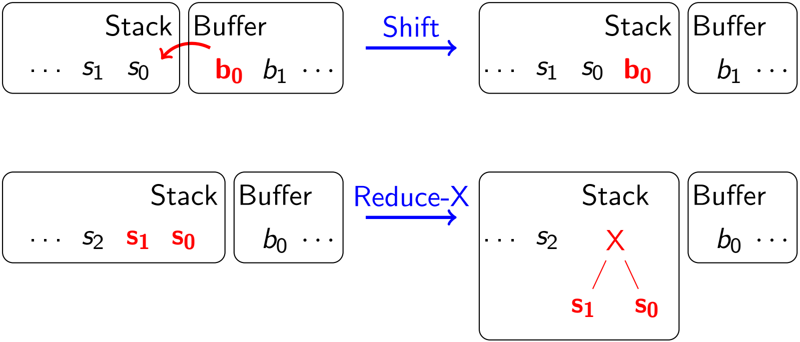
Outline

Transition-based Parsing: Standard Shift-Reduce

- Syntactic tree equivalent to a **sequence of actions**
- Classifier to predict actions
- Configuration = (Stack, Buffer)
  - Stack contains tree nodes
  - Buffer contains tokens
- Use actions to **derive new configurations** until the stack contains a single tree and the buffer is empty



Transition-based Parsing: Standard Shift-Reduce

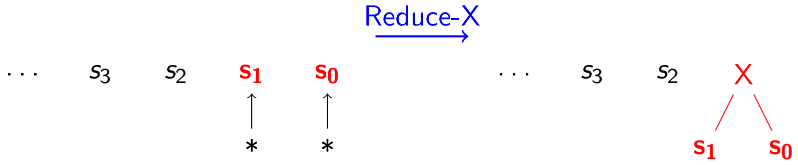


- s_i, b_i index elements in stack and buffer
- Reduce-Left-X, Reduce-Right-X for each non-terminal X
- Left/right: assign the head of the new constituent
  - useful because features use heads of constituents

Plan

Extending the Shift-Reduce algorithm

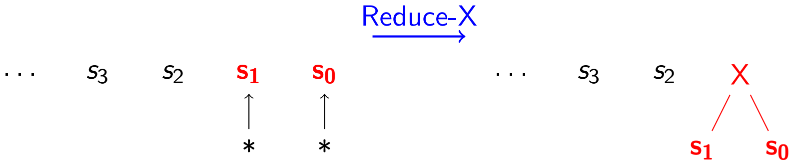
Standard shift-reduce: reductions apply to the 2 topmost elements in the stack:



$\rightarrow$ can only derive projective trees

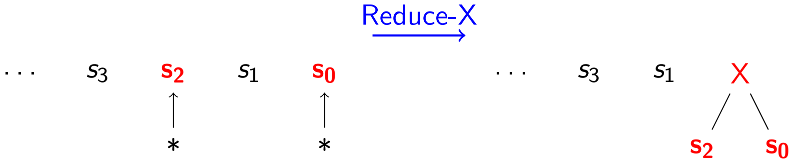
Extending the Shift-Reduce algorithm

Standard shift-reduce: reductions apply to the 2 topmost elements in the stack:



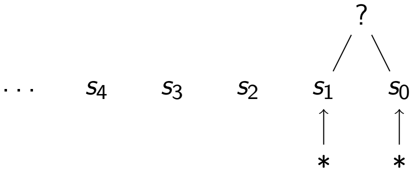
$\rightarrow$ can only derive projective trees

To handle **discontinuities**, allow reductions with s_0 and any other symbol in the stack. Side effect: **implicitly reordering terminals**



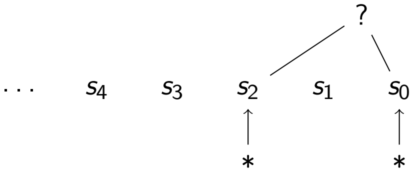
Shift-Reduce+Gap

- **GAP** action: access next non-terminal in the stack for a potential reduction
  - Choose dynamically which element in the stack is used for a reduction



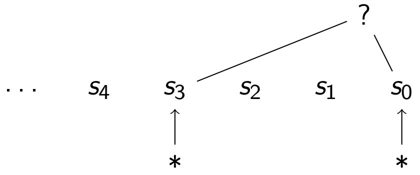
Shift-Reduce+Gap

- **GAP** action: access next non-terminal in the stack for a potential reduction
  - Choose dynamically which element in the stack is used for a reduction



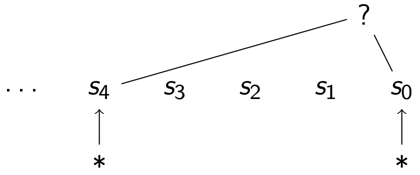
Shift-Reduce+Gap

- **GAP** action: access next non-terminal in the stack for a potential reduction
  - Choose dynamically which element in the stack is used for a reduction



Shift-Reduce+Gap

- **GAP** action: access next non-terminal in the stack for a potential reduction
  - Choose dynamically which element in the stack is used for a reduction



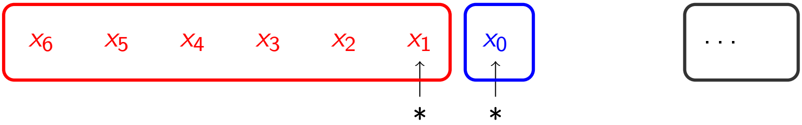
Shift-Reduce+Gap

The usual stack is split into 2 parts:

- A Stack S (bottom part)
- A Deque D (upper part)
- Reductions are always applied to s_0 and d_0
- (The buffer is still a buffer)

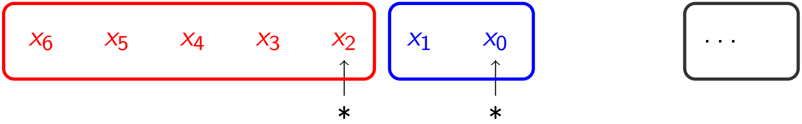
The GAP transition: Stack Deque Buffer

Projective case: s_0 and d_0 are the 2 topmost elements



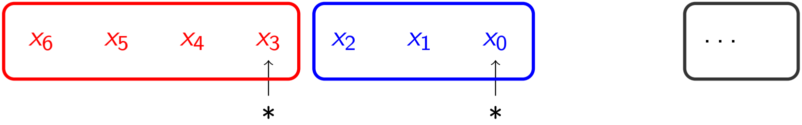
The GAP transition: Stack Deque Buffer

After 1 GAP



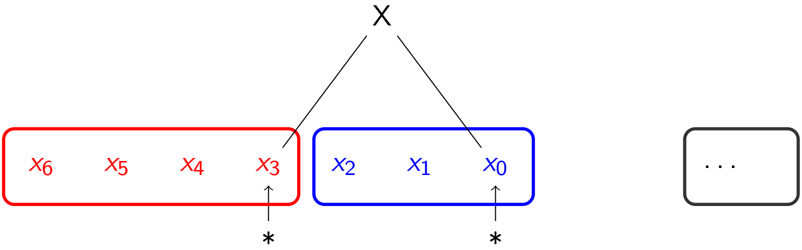
The GAP transition: Stack Deque Buffer

After 2 GAPs (Can gap as long as length of S is $>$ 1)



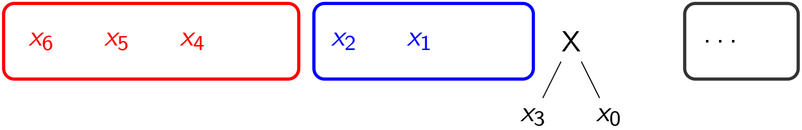
The GAP transition: Stack Deque Buffer

Reduction: pick s_0 and d_0



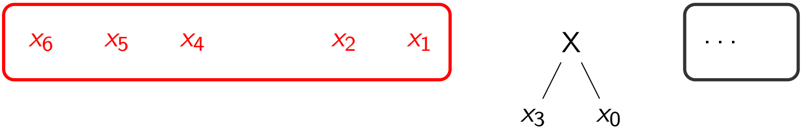
The GAP transition: Stack Deque Buffer

Reduction to X: create new node



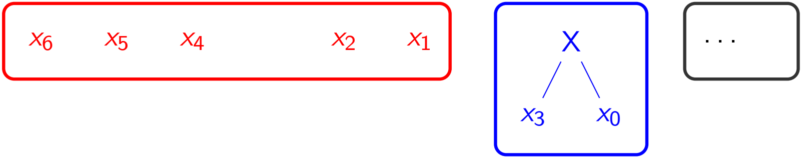
The GAP transition: Stack Deque Buffer

Reduction to X: flush D to S



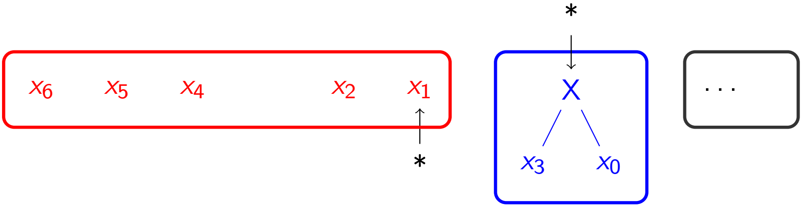
The GAP transition: Stack Deque Buffer

Reduction to X: push new node to D



The GAP transition: Stack Deque Buffer

Now, top of S is x_1 and top of D is X



Transition-based Parsing: Shift-Reduce+Gap

- Configuration = (Stack, **Deque**, Buffer)
  - Initial configuration = ($\emptyset$, $\emptyset$, $[w_1, w_2 \dots w_n]$)
  - Final configuration = ($\emptyset$, [A], $\emptyset$)
    - A = axiom

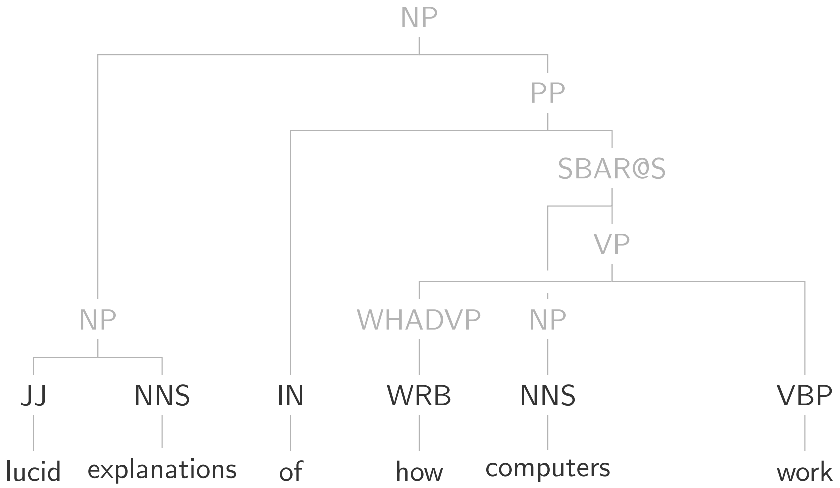
Transition set

	From	To
Shift	$(S, D, b_0 \vert B)$	$(S \vert D, [b_0], B)$
Reduce-Left/Right(X)	$(S \vert s_0, D \vert d_0, B)$	$(S \vert D, [X], B)$
Gap	$(S \vert s_0, D, B)$	$(S, s_0 \vert D, B)$

Let's see a full example ...

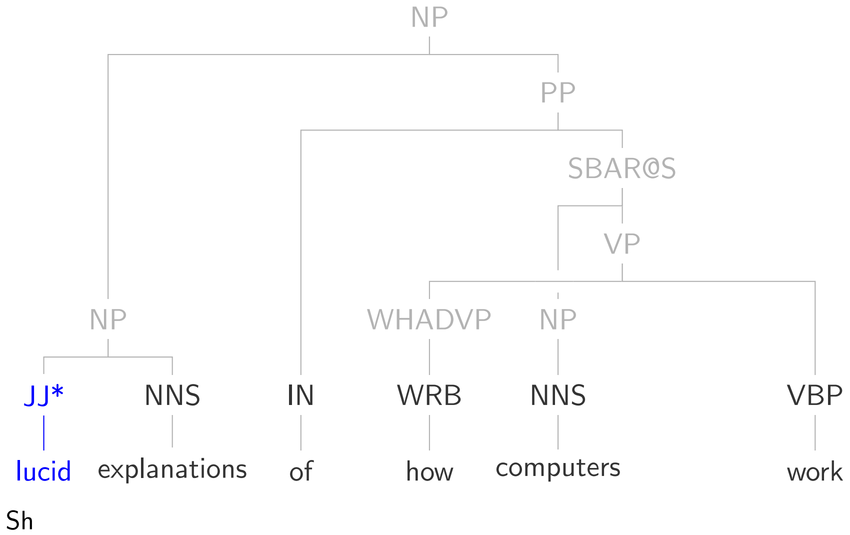
Shift-Reduce-Gap : Stack – Deque – Buffer

Initialisation [phrase from DPTB, Evang and Kallmeyer, 2011]



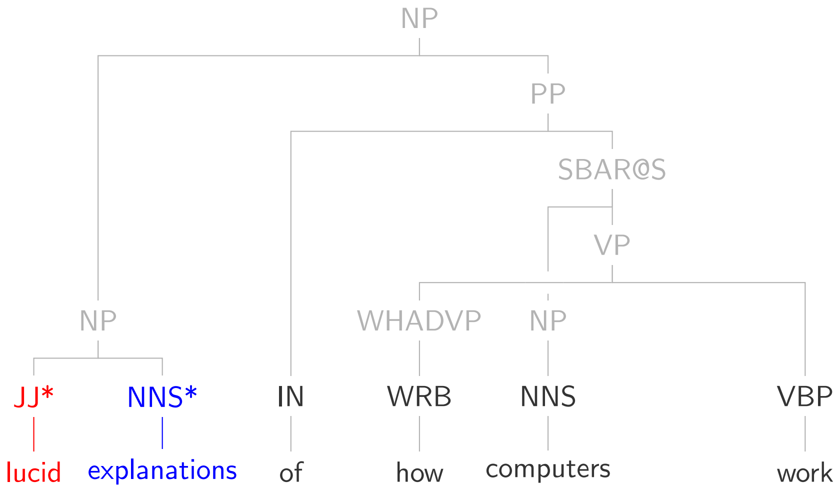
Shift-Reduce-Gap : Stack – Deque – Buffer

Shift



Shift-Reduce-Gap : Stack – Deque – Buffer

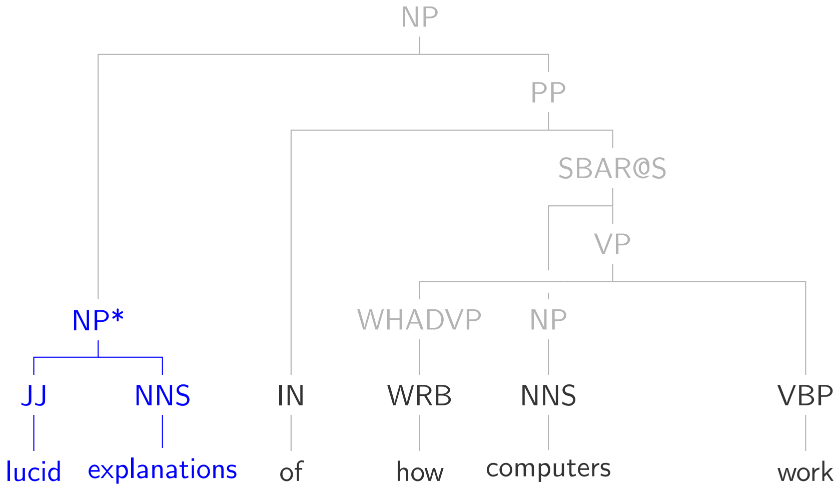
Shift



Sh, Sh

Shift-Reduce-Gap : Stack – Deque – Buffer

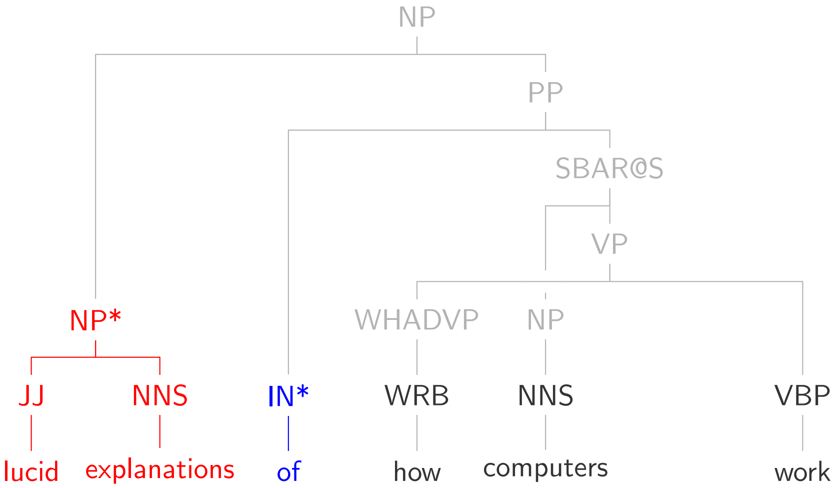
Reduce-NP



Sh, Sh, R(NP)

Shift-Reduce-Gap : Stack – Deque – Buffer

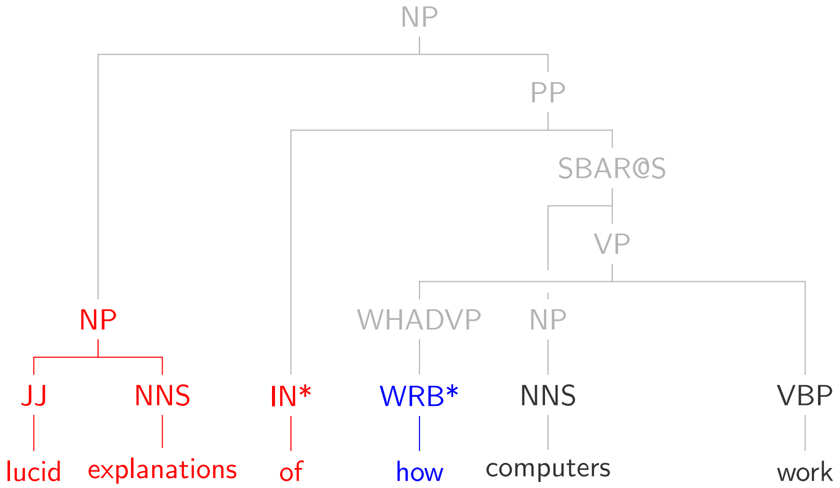
Shift



Sh, Sh, R(NP), Sh

Shift-Reduce-Gap : Stack – Deque – Buffer

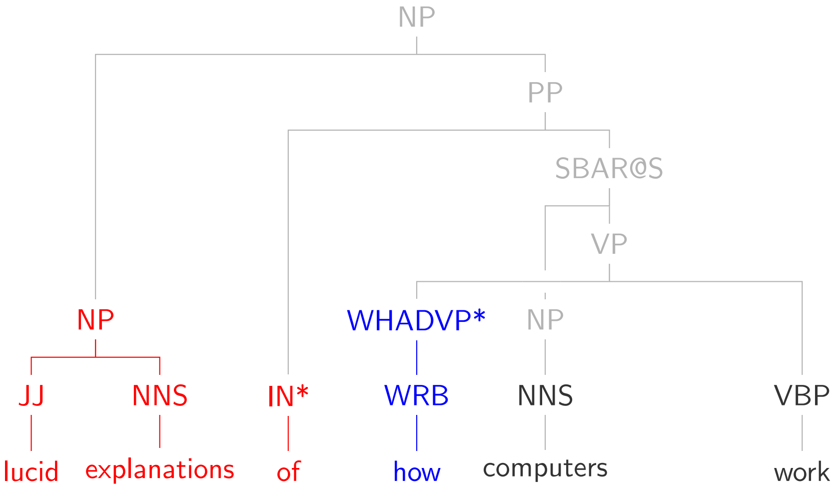
Shift



Sh, Sh, R(NP), Sh, Sh

Shift-Reduce-Gap : Stack – Deque – Buffer

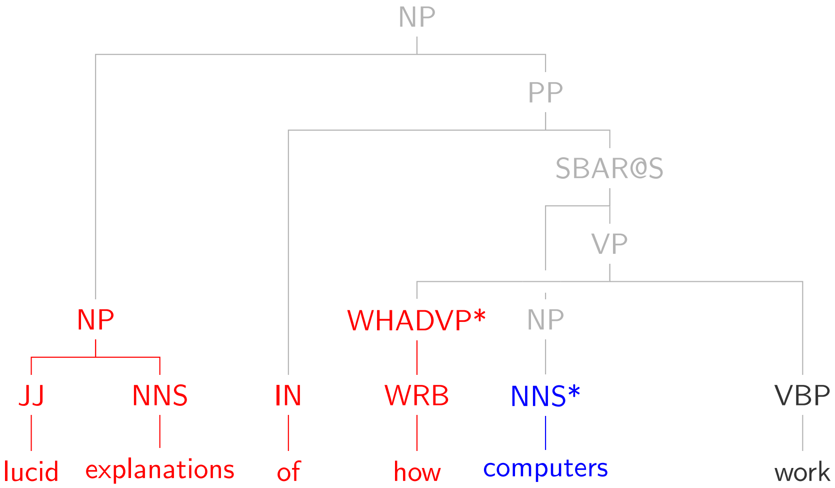
ReduceUnary-WHADVP



Sh, Sh, R(NP), Sh, Sh, RU(WHADVP)

Shift-Reduce-Gap : Stack – Deque – Buffer

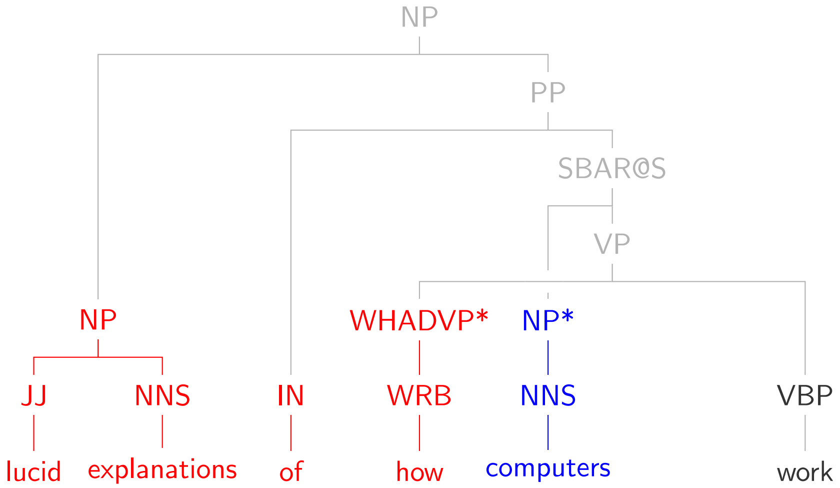
Shift



Sh, Sh, R(NP), Sh, Sh, RU(WHADVP), Sh

Shift-Reduce-Gap : Stack – Deque – Buffer

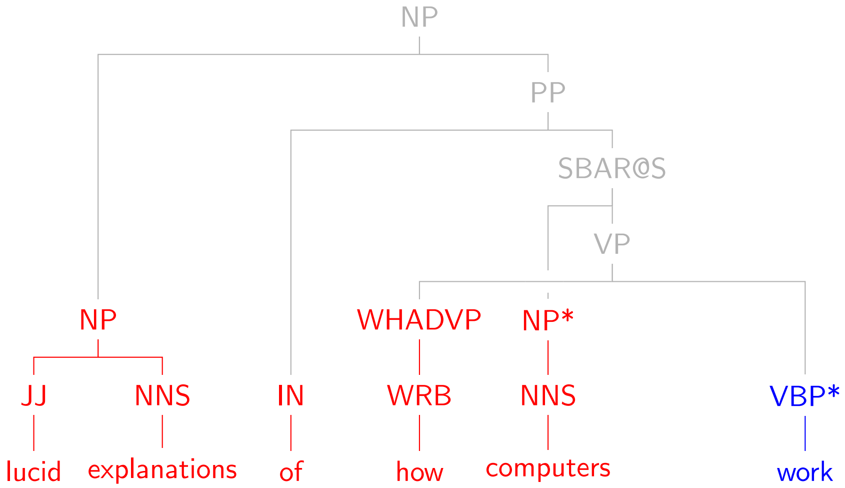
ReduceUnary-NP



Sh, Sh, R(NP), Sh, Sh, RU(WHADVP), Sh, RU(NP)

Shift-Reduce-Gap : Stack – Deque – Buffer

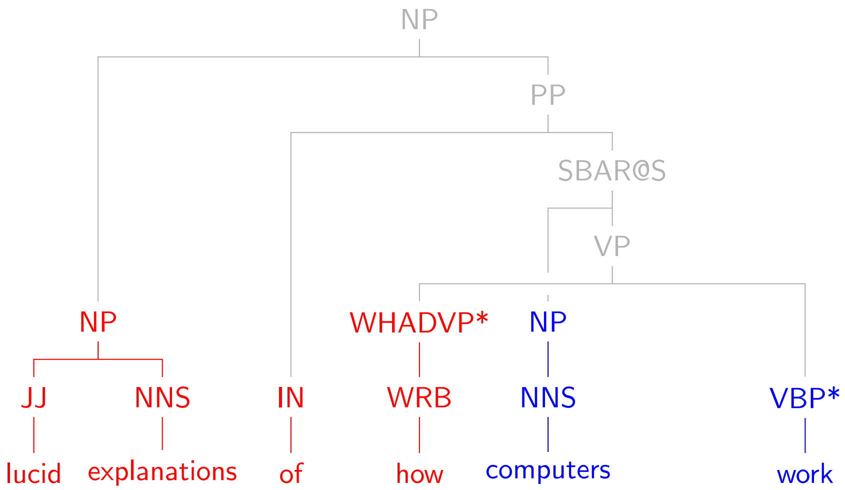
Shift



Sh, Sh, R(NP), Sh, Sh, RU(WHADVP), Sh, RU(NP), Sh

Shift-Reduce-Gap : Stack – Deque – Buffer

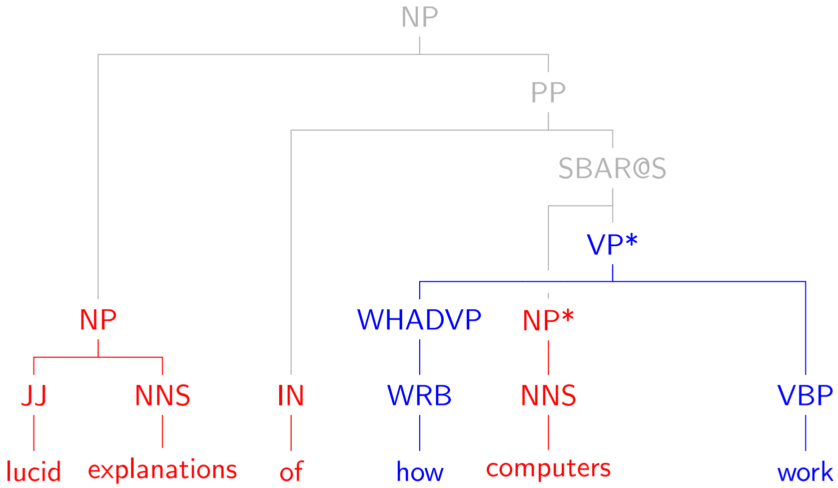
Gap



Sh, Sh, R(NP), Sh, Sh, RU(WHADVP), Sh, RU(NP), Sh, Gap

Shift-Reduce-Gap : Stack – Deque – Buffer

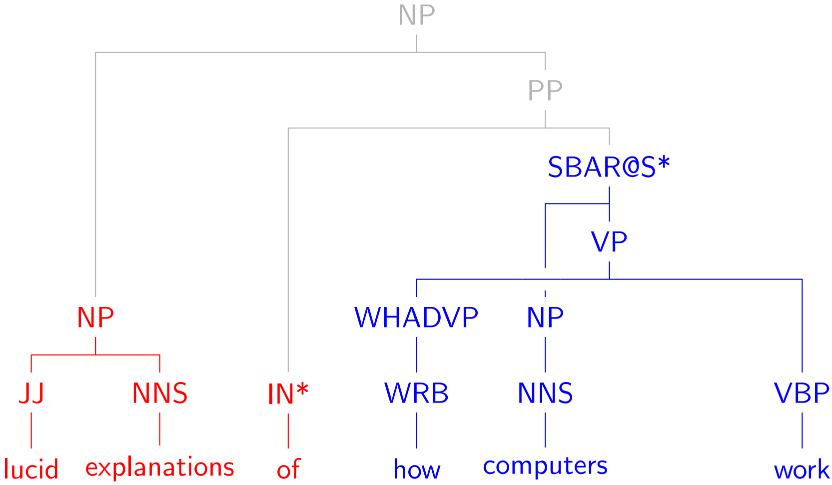
Reduce-VP



Sh, Sh, R(NP), Sh, Sh, RU(WHADVP), Sh, RU(NP), Sh, Gap, R(VP)

Shift-Reduce-Gap : Stack – Deque – Buffer

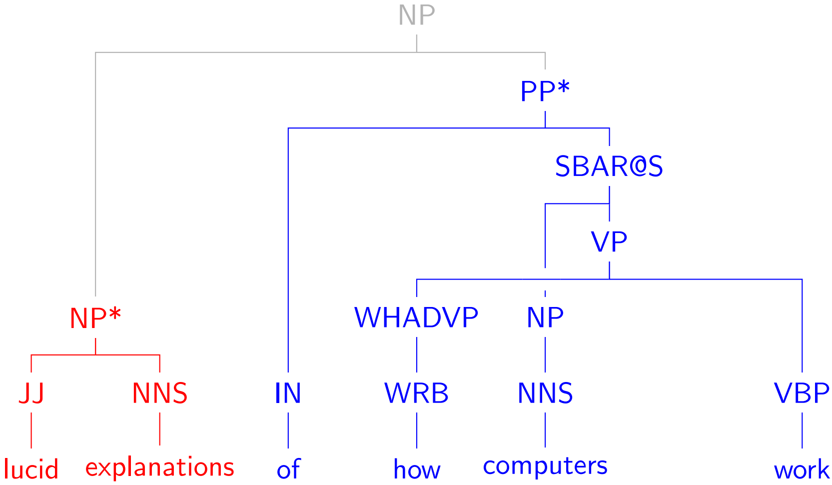
Reduce-S



Sh, Sh, R(NP), Sh, Sh, RU(WHADVP), Sh, RU(NP), Sh, Gap, R(VP), R(S)

Shift-Reduce-Gap : Stack – Deque – Buffer

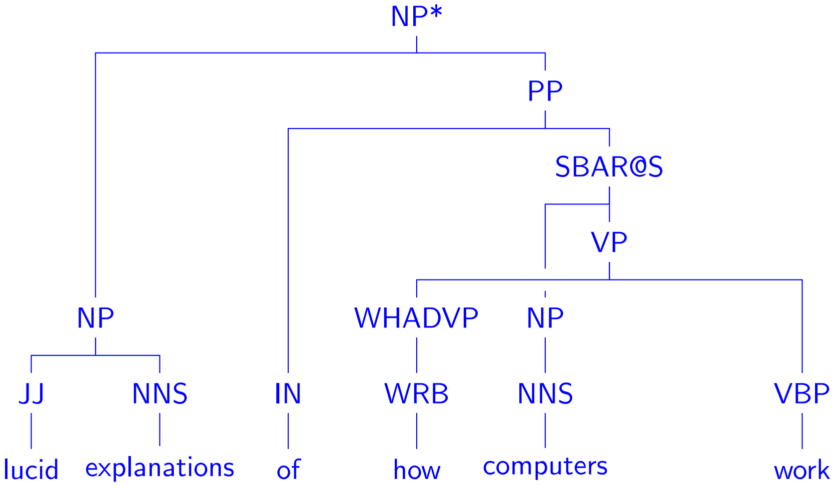
Reduce-PP



Sh, Sh, R(NP), Sh, Sh, RU(WHADVP), Sh, RU(NP), Sh, Gap, R(VP), R(S), R(PP)

Shift-Reduce-Gap : Stack – Deque – Buffer

Reduce-NP



Sh, Sh, R(NP), Sh, Sh, RU(WHADVP), Sh, RU(NP), Sh, Gap, R(VP), R(S), R(PP), R(NP)

Some properties of Shift-Reduce+Gap

- Derives any labelled discontinuous tree over a set of non-terminal symbols
  - handles well-nested and ill-nested trees
- With some (easily checked) constraints on actions: always outputs a tree
- Longest derivation for a sentence of size n is in $\mathcal{O}(n^2)$
  - In practice, parsing in linear time: limited number of discontinuities in datasets
- Related to Covington's (2001) non-projective dependency parsing algorithm
  - transition system with 3 data structures (Gómez-Rodríguez and Fernández-González, 2015)

Plan

Experiments: Data

- 2 German corpora:
  - Tiger Corpus (Brants et al., 2002), $\approx$ 50000 sentences
  - Negra Corpus (Skut et al., 1997), $\approx$ 20000 sentences
- Both were natively annotated with discontinuous constituents
- $\approx$ 30% of sentences contain at least one discontinuity
- Preprocessing
  - Head annotation with headrules
  - Head-outward binarization (+ order-0 Markovization)
  - Reattach punctuation locally (avoid spurious discontinuity)
- Assume tags are available (either gold or predicted)

Experiments: Classifier

- Deterministic oracle to transform gold trees to gold sequences of actions
- Simple averaged structured perceptron
  - Beam search
  - Early update
- Perceptron is biased towards longer derivations
  - padding derivations with IDLE actions (Zhu et al., 2013) to improve comparability between derivations in the beam
- C++ implementation : `github.com/mcoavoux/mtg`
  - Scalable to full corpora, 4700 tokens/s with beam size = 4
  - Tree structured stack (TSS) for compact representation of beam

Experiments: Feature templates

3 sets of feature templates

- **Baseline**, 40 templates: standard features for a projective constituency parser (Zhu et al., 2013)
- **+Extended**, 52 templates: adds information about
  - gapped elements (d_1, d_2)
  - extended context (s_3)
- **+Spans**, 87 templates: adds information about constituent boundaries (Hall et al., 2014)
  - e.g. leftmost/rightmost terminal spanned by s_0, etc..

Experiments: Results (Gold POS) – Internal comparisons

Disc. F1 : discontinuous constituents only. Evaluator: discodop (Van Cranenburgh et al., 2016)

Beam size	TigerHN8 (dev)	
Gap, **+Spans**	F1	Disc. F1
2	81.86	48.49
4	83.27	53.00
8	83.61	54.42
16	83.84	54.81
32	**84.32**	**56.22**
64	84.14	56.01
128	84.05	55.76

- Beam size helpful for disc. constituents. From 2 to 32:
  - + 8 for discontinuous constituents
  - + 3 for all constituents
  - Search compensates for lack of global view

Experiments: Results (Gold POS) – Internal comparisons

Test	TigerHN08		Improvement over baseline	
Beam = 4	F1	Disc. F1	F1	Disc. F1
Shift-Reduce+Gap, **Baseline**	81.67	44.83		
Shift-Reduce+Gap, **+Extended**	82.43	48.81	+ 0.7	+ 4.0
Shift-Reduce+Gap, **+Spans**	**83.16**	**49.76**	+ 1.4	+ 4.9

- **+Extended**: information about content of gap is useful especially for discontinuities
- **+Spans**: information about constituent boundaries is useful

Experiments: Results (Gold POS) – External comparisons

Comparison with chart parsers

- CKY-like decoding
- Kallmeyer and Maier (2013): Probabilistic LCFRS
- Van Cranenburgh et al. (2016): DOP model

	Negra sentence length	F1
Kallmeyer and Maier, 2013	≤ 30	75.8
Van Cranenburgh et al., 2016	≤ 40	76.8
Shift-Reduce+Gap, beam=32	All	**82.16**

Experiments: Results (Gold POS) – External comparisons

Comparison with transition based parsers

- Maier 2015: extends shift-reduce with swap action
  - Swap: push 2nd element of stack back onto the buffer (adapted from d-parsing)
  - Same setting as ours: global perceptron with beam search

		Tiger	
	beam size	F1	Disc. F1
Maier 2015	4	74.71	18.8
Maier and Lichte 2016	4	76.46	16.3
Shift-Reduce+Gap	4	80.40	46.5
Shift-Reduce+Gap	32	**81.60**	**49.2**

Experiments: Results (Gold POS) – External comparisons

Comparison with dependency parsing based methods

Fernández-González and Martins (2015) *Parsing as Reduction*

1. Convert trees to (non-projective) dependency trees
2. Parse with dependency parser
3. Convert result to discontinuous constituency trees

	Gold POS			Predicted POS
	Negra	TigerHN8	TigerSPMRL	TigerSPMRL
FeMa2015	80.5	**84.2**	80.6	77.3
SR+Gap	**82.2**	84.0	**81.5**	**79.3**

Plan

Discussion: Gap vs Swap

Swap and Gap

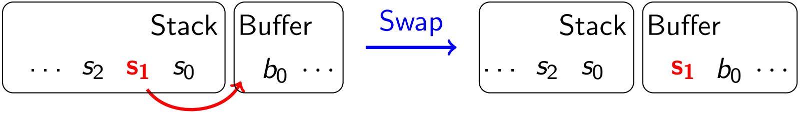
- both extensions of shift-reduce
- both $\mathcal{O}(n^2)$
- but rather large difference in accuracies (in comparable experimental settings).

Why?

- Main hypothesis: Shift-Reduce+Gap tends to produce much shorter derivations (easier to learn)

Discussion: Gap vs Swap

Swap action: push the second element of the stack back onto the buffer. (It must be a terminal)



Swap transition system (Maier, 2015) uses 2 usual data structures (stack + buffer).

Discussion: Derivation Length

We expect the Gap transition system to produce **shorter derivation** than Swap in general.

1. Each swapped terminal must be shifted again (and might be reswapped and reshifted if need be)
2. Both systems are implicitly predicting a reordering of terminals that would make a tree projective
   - Swap **terminals** only → reordering terminals not efficient
   - Can gap **whole subtrees** → reordering more efficient

Discussion: Derivation Length

Some measures on the Tiger corpus (train) using an oracle:

	SR+Gap	SR+Swap
Average derivation length wrt n	$2.03n$	$3.09n$
Longest derivation	276	2187
Total number of gaps/swaps	64096	411970
Max consecutive gaps/swaps	10	69

Conclusion

Summary

- New transition system for efficient and accurate discontinuous parsing
- State-of-the-art results on German treebanks

Perspectives

- Application to other languages (English, French) and other types of linguistic discontinuities (MWE?)
- bi-LSTM encoder (Cross and Huang, 2016)
- Dynamic oracle?
- Relationship to LCFRS automata?

https://github.com/mcoavoux/mtg/
mcoavoux@linguist.univ-paris-diderot.fr

Thanks!

Comments? Questions?

Thanks to Chloé Braud, Héctor Martínez Alonso, Djamé Seddah, Olga Seminck

Transition System

Input	$t_1[w_1]t_2[w_2]\dots t_n[w_n]$
Axiom	$\langle \epsilon, \epsilon, t_1[w_1]t_2[w_2]\dots t_n[w_n] \rangle$
Goal	$\langle \epsilon, S[w], \epsilon \rangle$
SHIFT	$\dfrac{\langle S, D, t[w] \mid B \rangle}{\langle S \mid D, t[w], B \rangle}$
REDUCE-UNARY(X)	$\dfrac{\langle S, d_0[h], B \rangle}{\langle S, X[h], B \rangle}$
REDUCE-RIGHT(X)	$\dfrac{\langle S \mid s_0[h], D \mid d_0[h'], B \rangle}{\langle S \mid D, X[h'], B \rangle}$
REDUCE-LEFT(X)	$\dfrac{\langle S \mid s_0[h], D \mid d_0[h'], B \rangle}{\langle S \mid D, X[h], B \rangle}$
GAP	$\dfrac{\langle S \mid s_0[h], D, B \rangle}{\langle S, s_0[h] \mid D, B \rangle}$

Experiments: Results with predicted pos

TigerM15	F1 (spmrl.prm) ≤ 70	All
Versley (2014), EasyFirst	73.90	-
Fernández-González and Martins (2015)	77.72	77.32
sr-gap, beam=32, +Spans	79.44	79.26
sr-gap, greedy, bi-lstms, own tagging		**82.22**

- + 3 over structured perceptron

Relationship to dependency parsing

Shift-Reduce+Gap is related to Covington's (2001) (family of) algorithm(s) for unrestricted dependency parsing

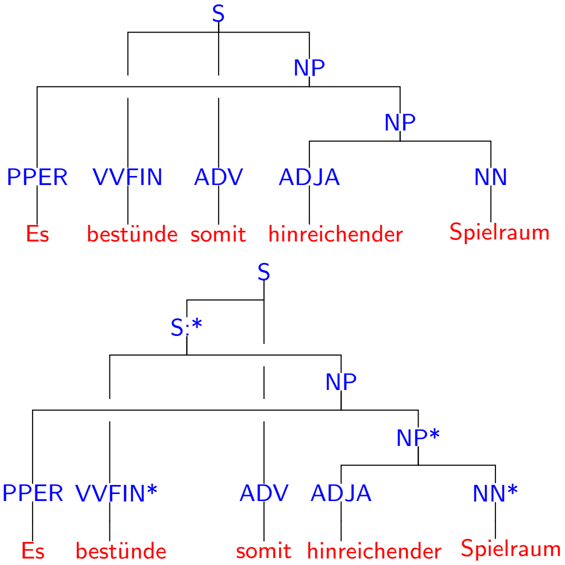
Algorithm 1 Covington's algorithm for d-parsing $\mathcal{O}(n^2)$

```
for i = 1 to n do
    for j = i − 1 downto 1 do
        link(i,j)
    end for
end for
```

Link(i,j): either add arc $i \rightarrow j$, or arc $j \rightarrow i$ or do nothing.

- Can be formulated as a transition system with 3 data structures

Binarization (sentence from Tiger)



* indicates head percolation