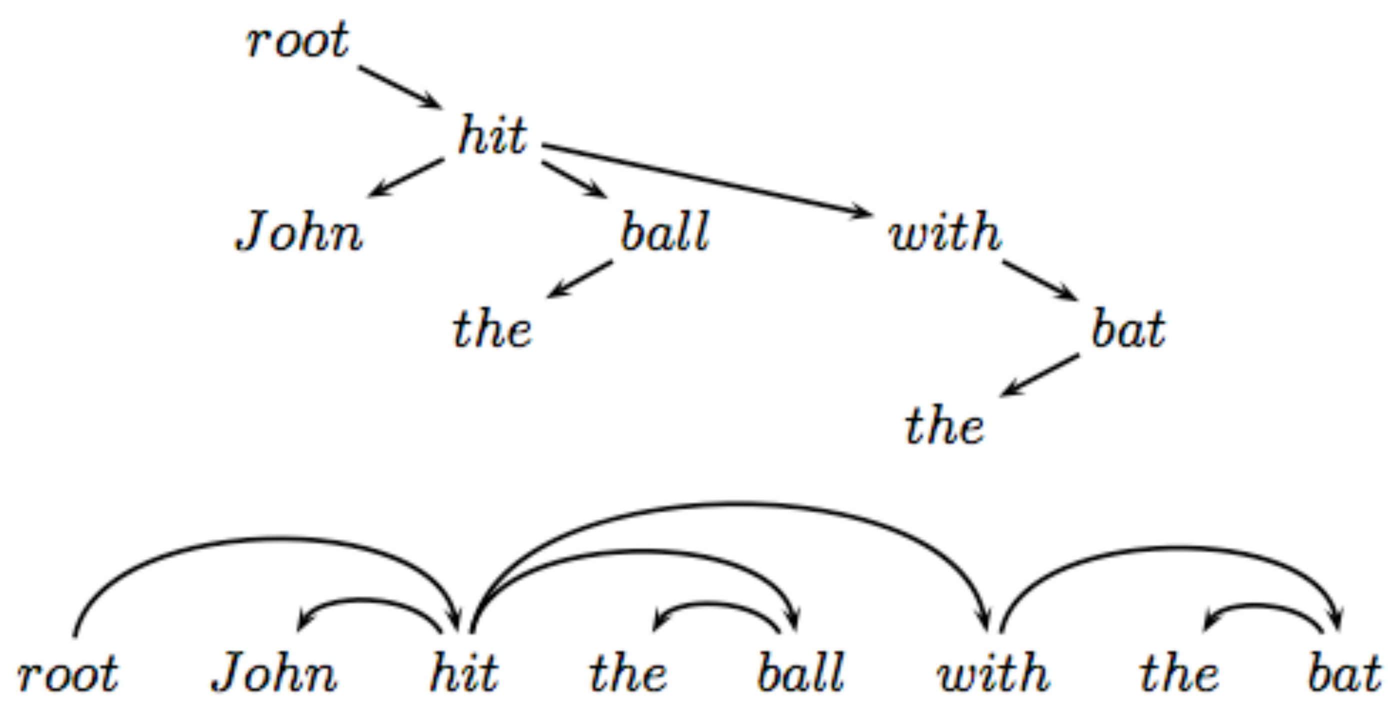
Introduction to Syntax and Parsing
ACS 2015/16

Stephen Clark

L3: Graph-Based Dependency Parsing

UNIVERSITY OF CAMBRIDGE

Untyped Dependency Trees

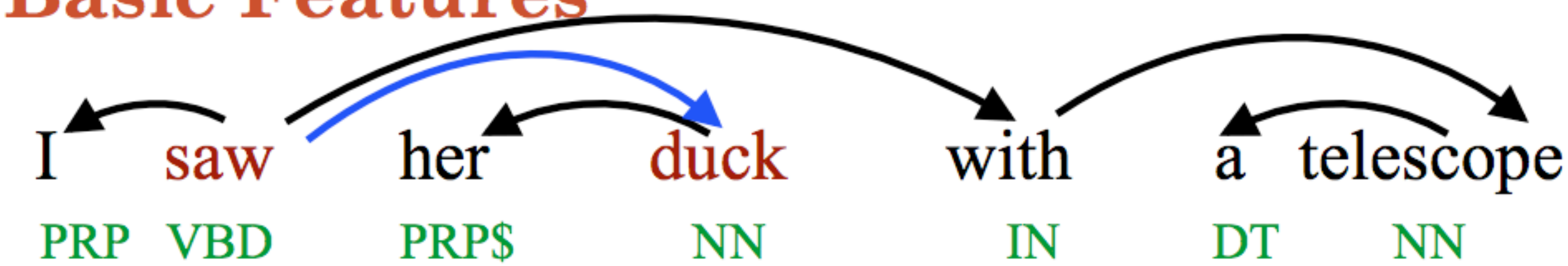


Taken from McDonald et al.

A tree is projective iff an edge from word w to word u implies that w is an ancestor of all words between w and u

Edge-Based Linear Model

Basic Features



Uni-gram features	Saw_VBD, saw, VBD duck_NN, duck, NN
Bi-gram features	saw_VBD_duck_NN, VBD_duck_NN, saw_duck_NN, saw_VBD_NN, saw_VBD_duck, Saw_duck, VBD_NN
In between POS features	VBD_PRP$_NN
Surrounding word POS features	VBD_PRP$_PRP$_NN, PRP_VBD_PRP$_NN, VBD_PRP$_NN_IN, PRP_VBD_NN_IN

taken from Wang and Zhang, NAACL tutorial 2010

$$score(x_i \rightarrow x_j) = \sum_k \lambda_k \cdot f_k(x_i \rightarrow x_j)$$

UNIVERSITY OF CAMBRIDGE

Dependency Parsing Formally

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{(i,j)\in \boldsymbol{y}} s(i, j) = \sum_{(i,j)\in \boldsymbol{y}} \mathbf{w} \cdot \mathbf{f}(i, j)$$

x is a sentence, ***y*** is a tree

(i,j) is an edge from *i*th word to *j*th word

s is the scoring function

f is the feature function, **w** is the weight vector

UNIVERSITY OF CAMBRIDGE

Maximum Spanning Trees

Assume we know the weight vector, **w**

Consider the following directed graph for sentence ***x***:

$G_{\boldsymbol{x}} = (V_{\boldsymbol{x}}, E_{\boldsymbol{x}})$ where

$$V_{\boldsymbol{x}} = \{x_0 = \text{root}, x_1, \ldots, x_n\} \text{ and}$$
$$E_{\boldsymbol{x}} = \{(i,j) : x_i \neq x_j, x_i \in V_{\boldsymbol{x}}, x_j \in V_{\boldsymbol{x}} - \text{root}\}$$

The highest-scoring (projective) dependency tree is equivalent to the (projective) *maximum spanning tree*

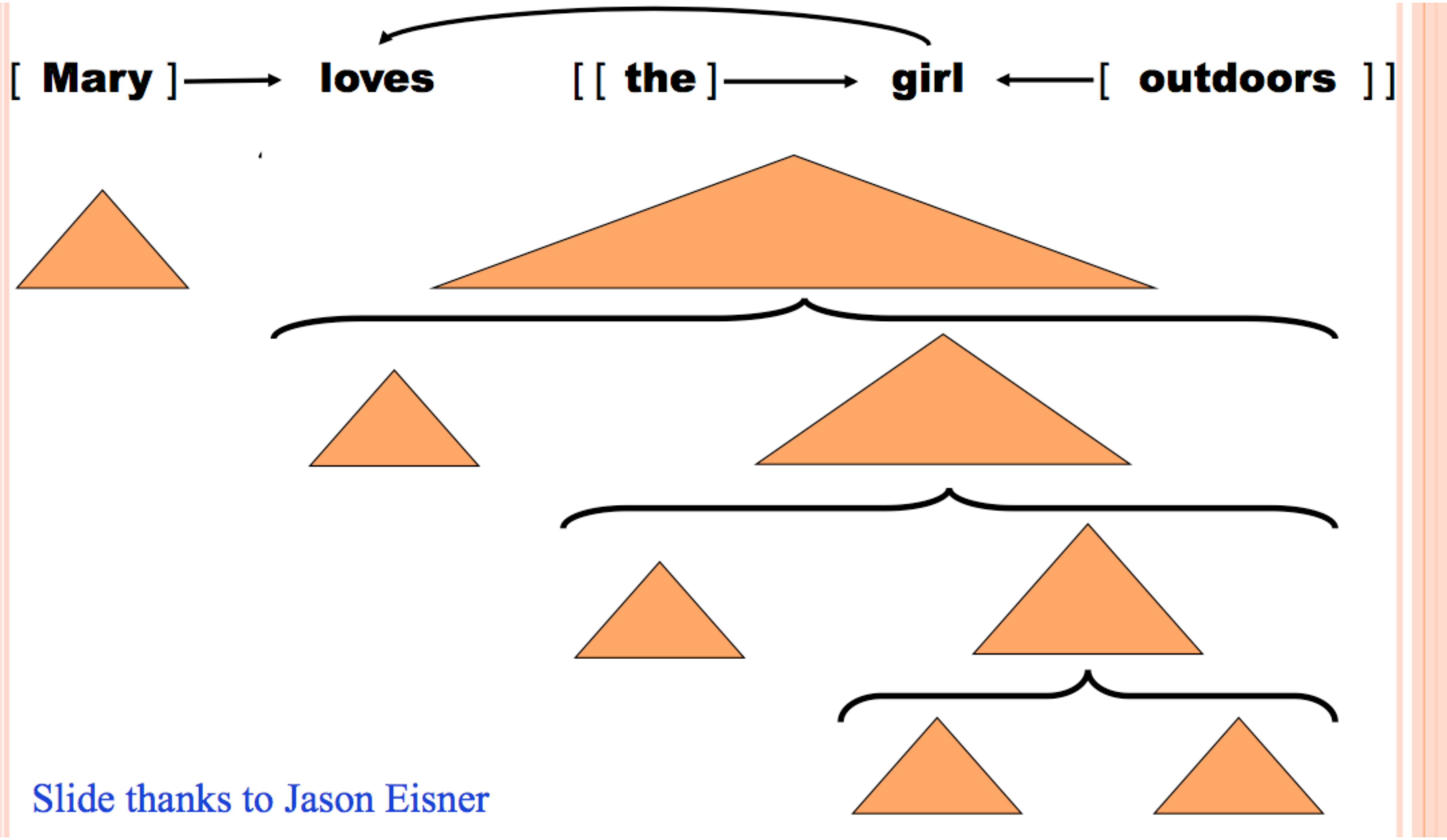
Decoding: finding the MST

The Chu-Liu-Edmonds algorithm (1965,67) finds the MST for *non-projective* trees; there is an O(n^2) implementation

For projective trees, the CKY algorithm can be adapted for dependency parsing to give an O(n^5) algorithm

There is a clever alternative chart-based algorithm from Eisner (1996) which runs in O(n^3)

CKY-style Dependency Parsing



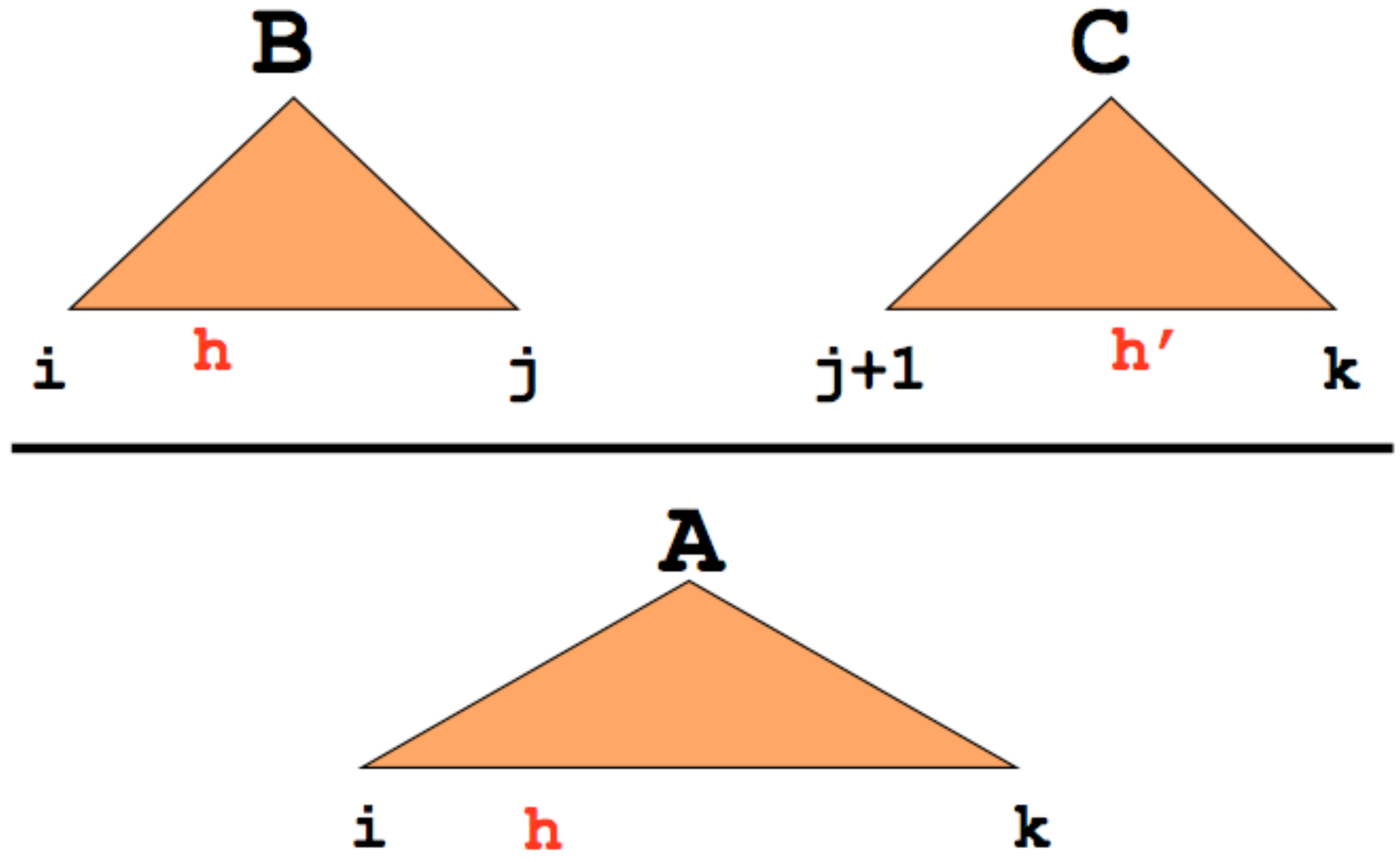
Slide thanks to Jason Eisner

UNIVERSITY OF CAMBRIDGE

Why CKY is O(n^5) not O(n^3)

... advocate visiting relatives

... hug visiting relatives



~~O(n^3 combinations)~~

O(n^5 combinations)

Slide thanks to Jason Eisner

Dependency Parsing Algorithms

Name	Inventor	Projectivity	Complexity
CKY-style chart parsing	Cocke–Younger–Kasami	Projective	$O(n^5)$
Eisner $O(n^3)$ parsing alg.	Eisner (96)	Projective	$O(n^3)$
Maximum Spanning Tree	Chu-Liu-Edmonds (65, 67)	Non-projective	$O(n^2)$
Shift-Reduce style parsing	Yamada, Nivre	Projective	$O(n)$

taken from Wang and Zhang, NAACL tutorial 2010

Shift-Reduce Dependency Parsing

- S – Shift
- R – Reduce
- AL – ArcLeft
- AR – ArcRight

He does it here

taken from Wang and Zhang, NAACL tutorial 2010

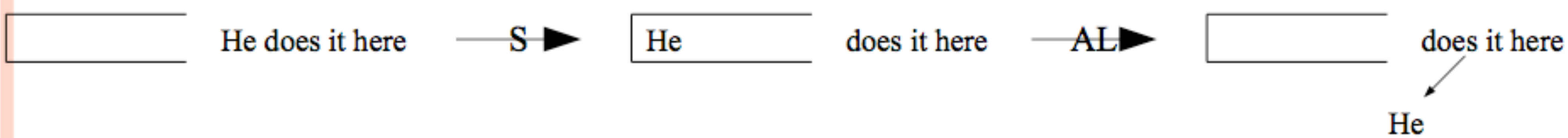
Shift-Reduce Dependency Parsing

- S – Shift
- R – Reduce
- AL – ArcLeft
- AR – ArcRight

$[\ ]$ He does it here $\xrightarrow{S}$ $[\text{He}]$ does it here

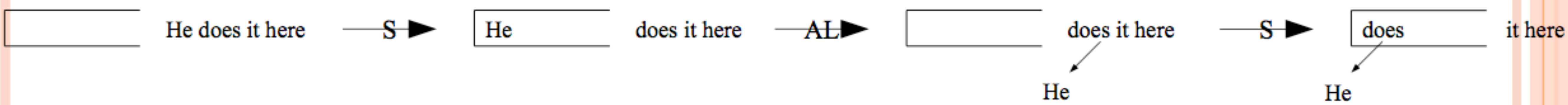
Shift-Reduce Dependency Parsing

- S – Shift
- R – Reduce
- AL – ArcLeft
- AR – ArcRight



Shift-Reduce Dependency Parsing

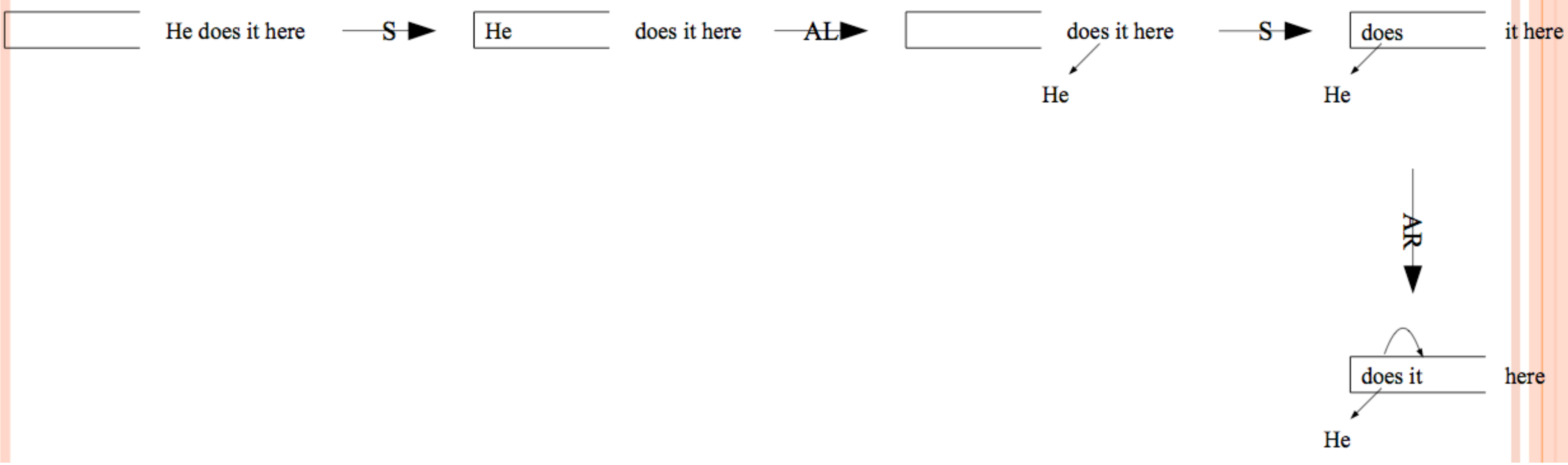
- S – Shift
- R – Reduce
- AL – ArcLeft
- AR – ArcRight



UNIVERSITY OF CAMBRIDGE

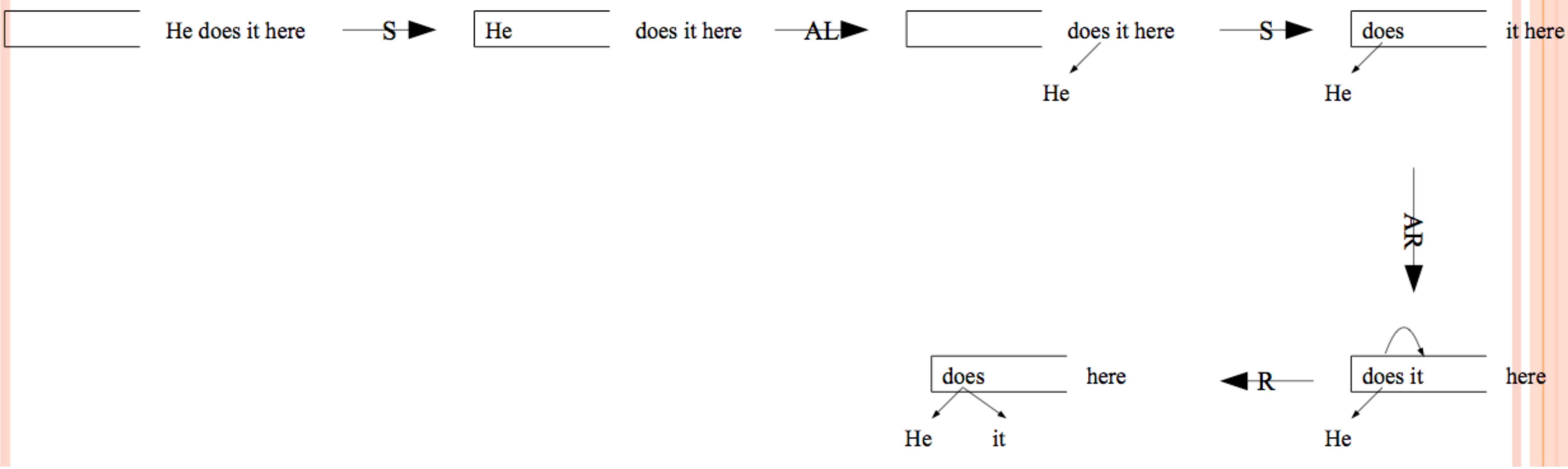
Shift-Reduce Dependency Parsing

- S – Shift
- R – Reduce
- AL – ArcLeft
- AR – ArcRight



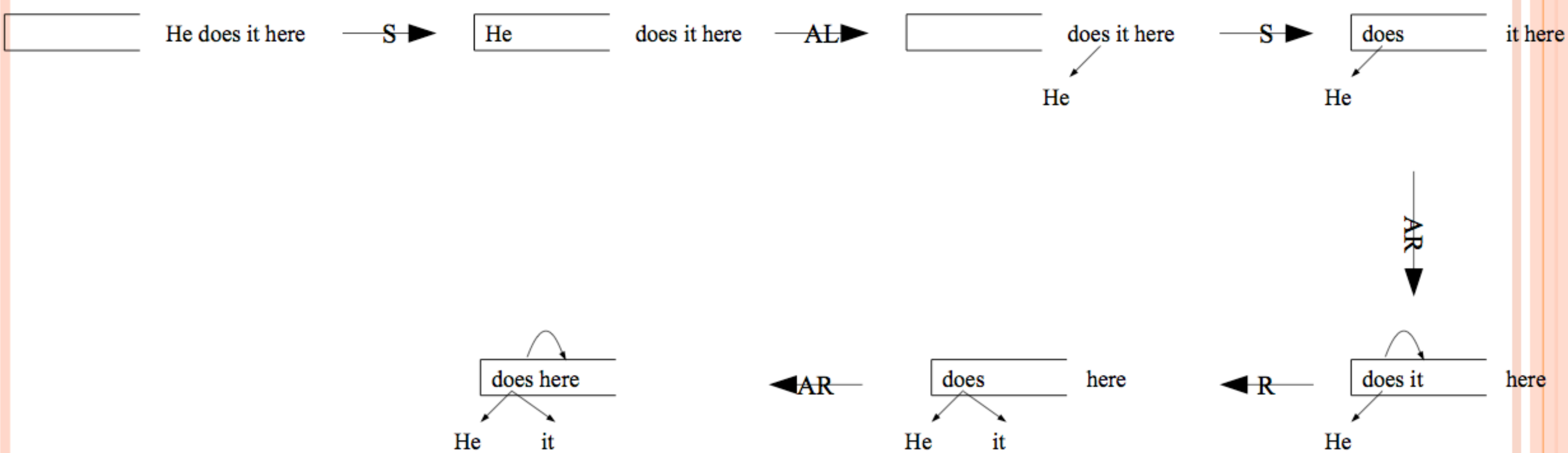
Shift-Reduce Dependency Parsing

- S – Shift
- R – Reduce
- AL – ArcLeft
- AR – ArcRight



Shift-Reduce Dependency Parsing

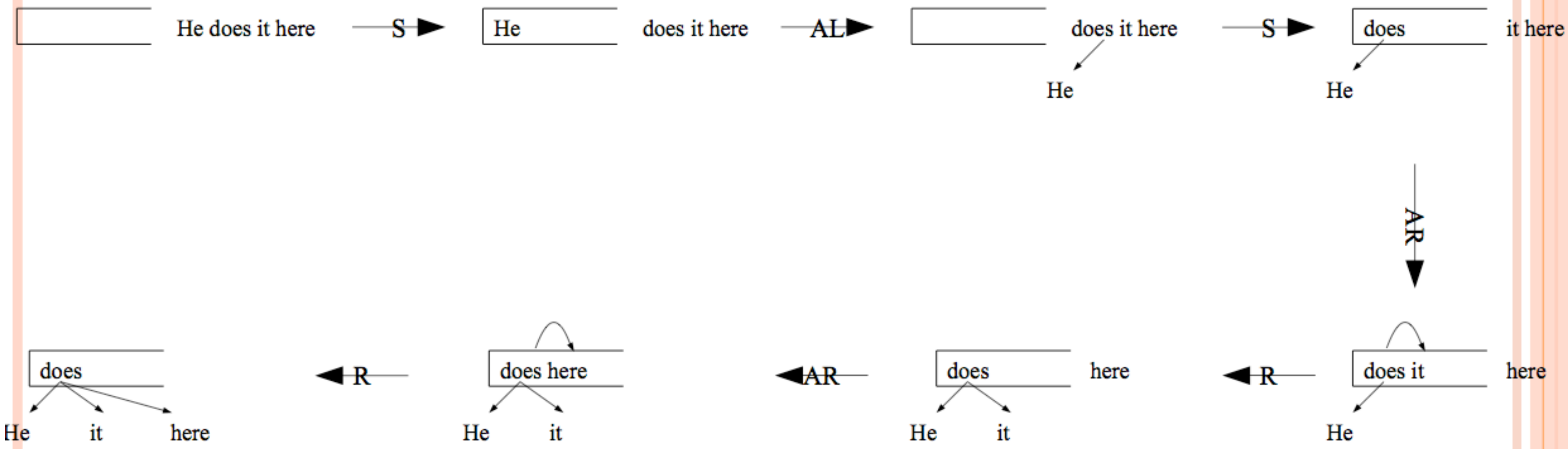
- S – Shift
- R – Reduce
- AL – ArcLeft
- AR – ArcRight



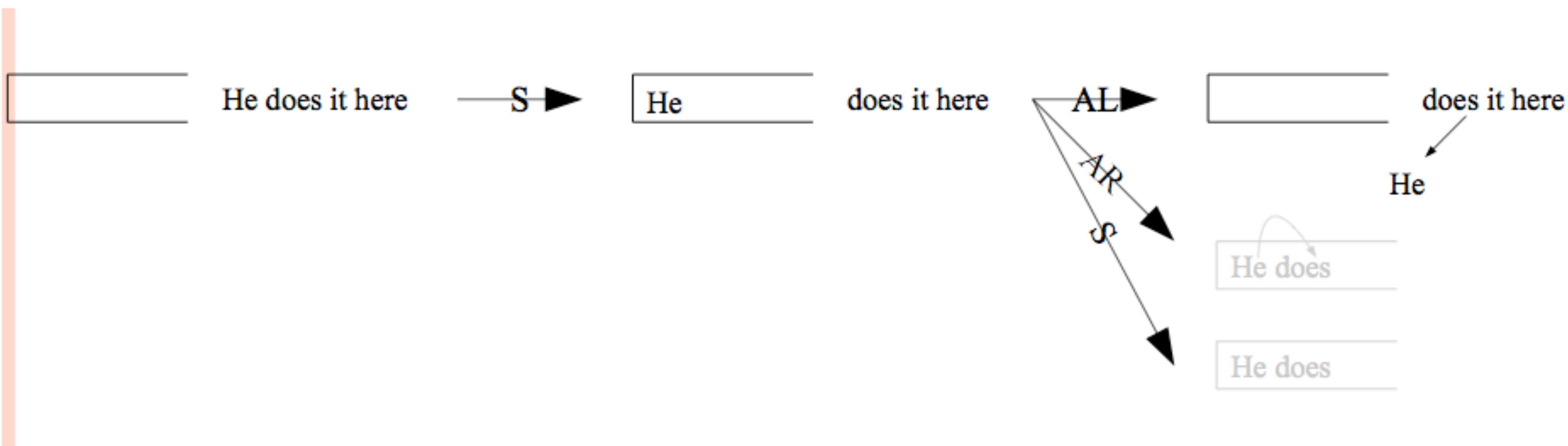
UNIVERSITY OF CAMBRIDGE

Shift-Reduce Dependency Parsing

- S – Shift
- R – Reduce
- AL – ArcLeft
- AR – ArcRight



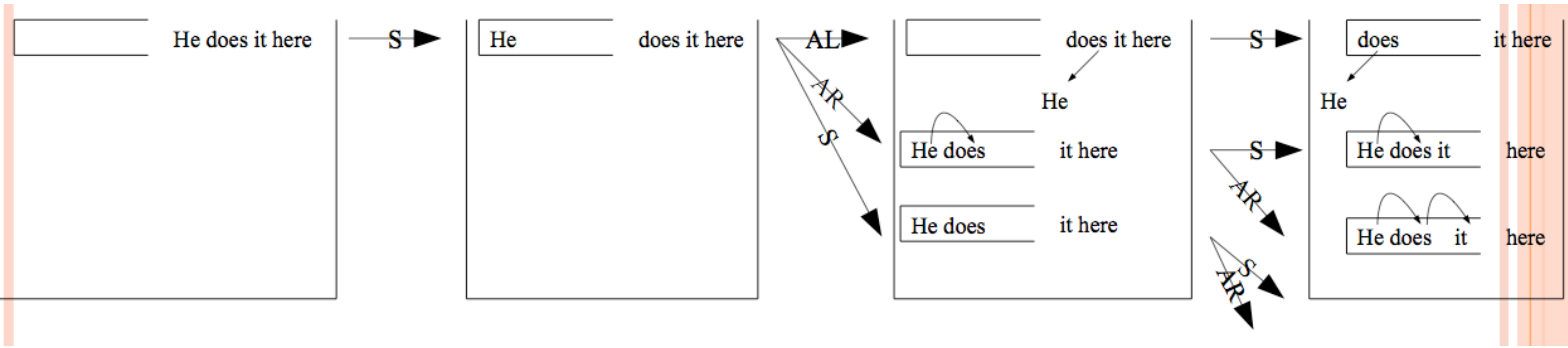
Greedy Local Search



taken from Wang and Zhang, NAACL tutorial 2010

Suffers from search errors, but potentially very fast (linear time)

Beam Search



Suffers from fewer search errors, but less fast (still linear time)