# Dependency Structures

Computational Linguistics

Emory University

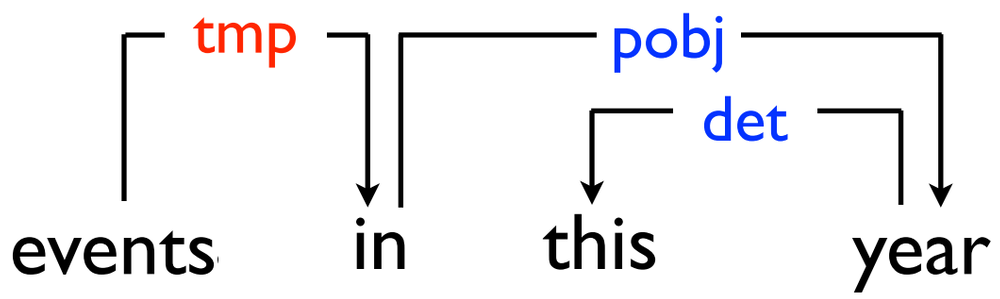
Jinho D. Choi

EMORY
UNIVERSITY

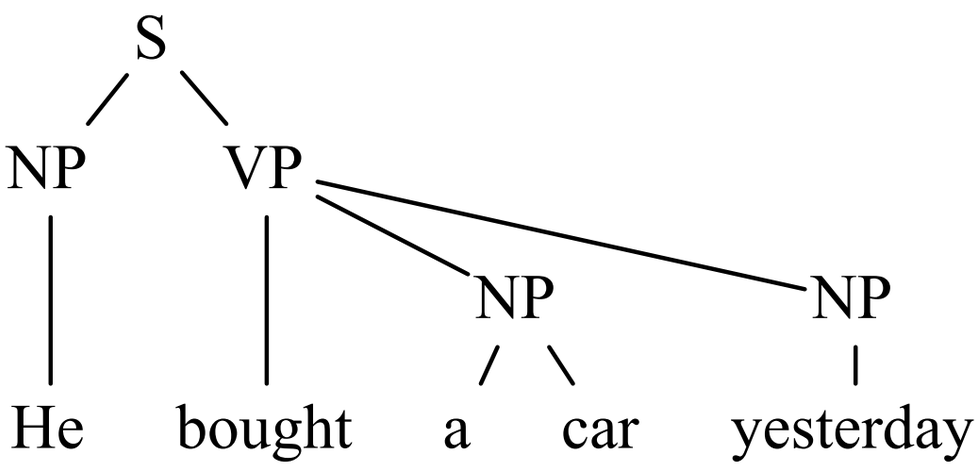
# Dependency Structures

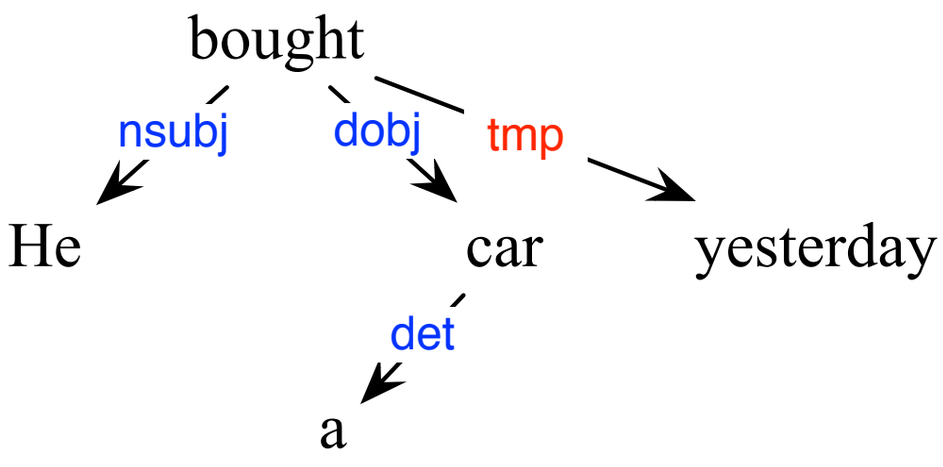
A syntactic or semantic (or other) relation between a pair of tokens.

dependency




## Phrase structures vs. Dependency structures



# Dependency Structures

## Phrase structures

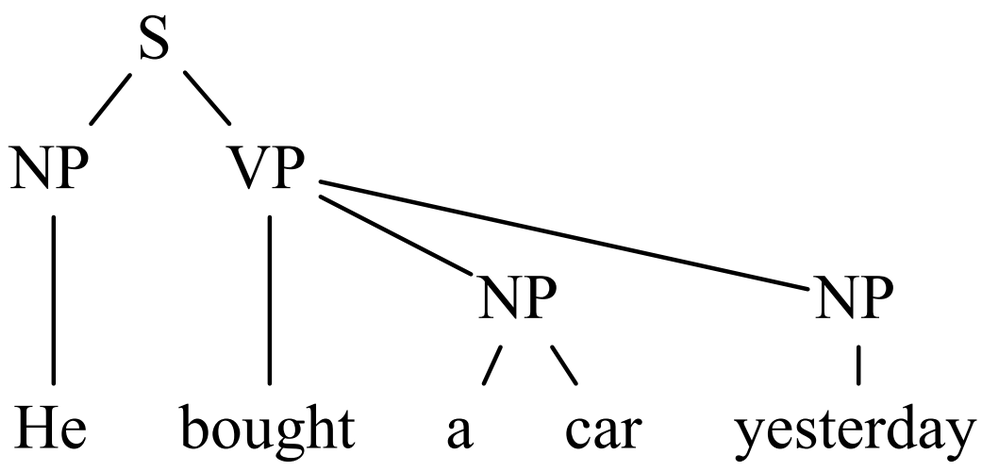
Starts with the bottom level phrases (tokens).

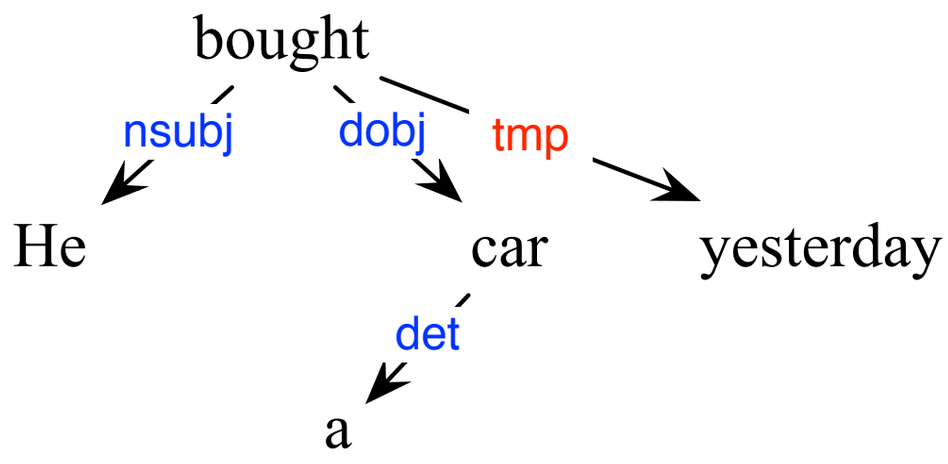
Group smaller phrases into bigger phrases.

## Dependency structures

Starts with vertices (tokens).

Build a graph by adding edges between vertices (arcs).



# Dependency Graph

For a sentence $s = w_1 \dots w_n$, a dependency graph $G_s = (V_s, A_s)$

$$V_s = \{w_0 = \textit{root}, w_1, \dots, w_n\}$$

$$A_s = \{(w_i, w_j, r) : i \neq j, w_i \in V_s, w_j \in V_s - \{w_0\}, r \in R_s\}$$

set of dependency relations

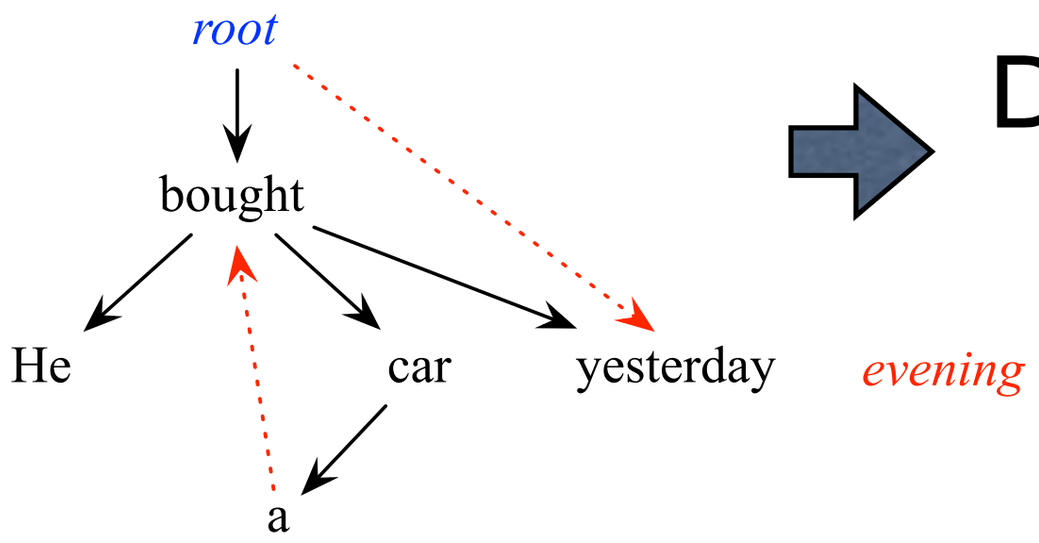
## A well-formed dependency graph

Root

Single head

Connected

Acyclic



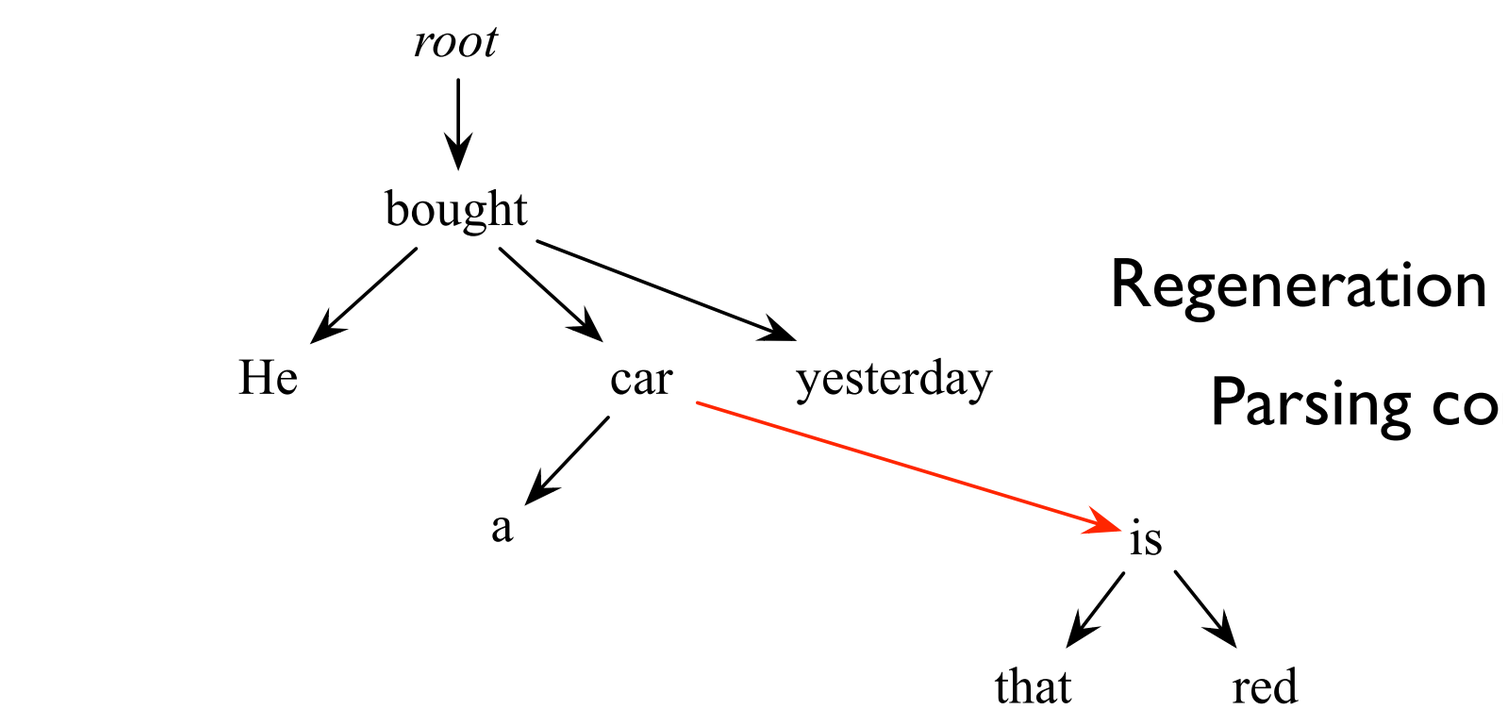

Dependency Tree

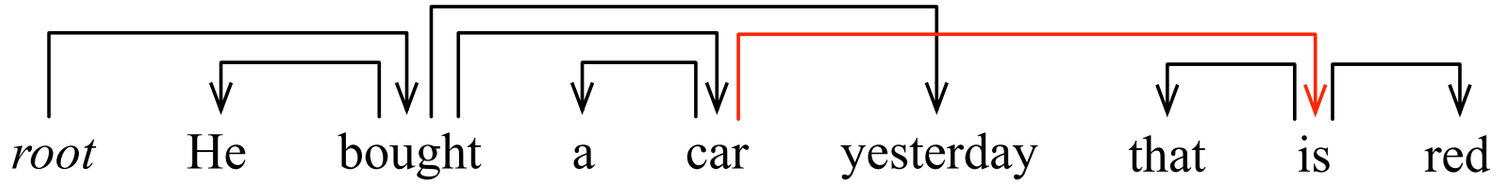
# Dependency Graph

## Projectivity

A projective dependency tree has no crossing arc when all vertices are lined up in linear order.

*He bought a car yesterday that is red*




Advantages:

Regeneration of the original sentence.

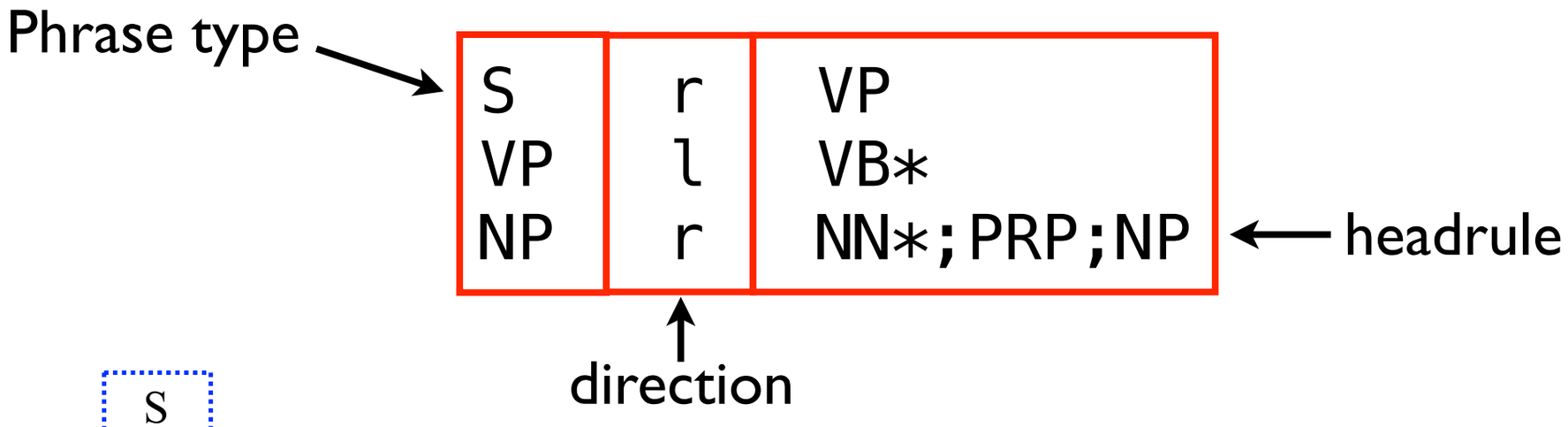
Parsing complexity: O($n$) vs. O($n^2$).

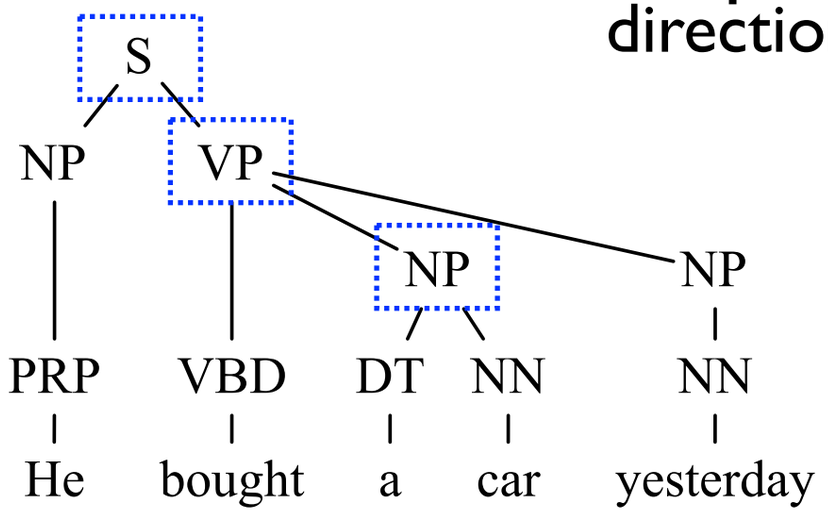
# Phrase To Dependency

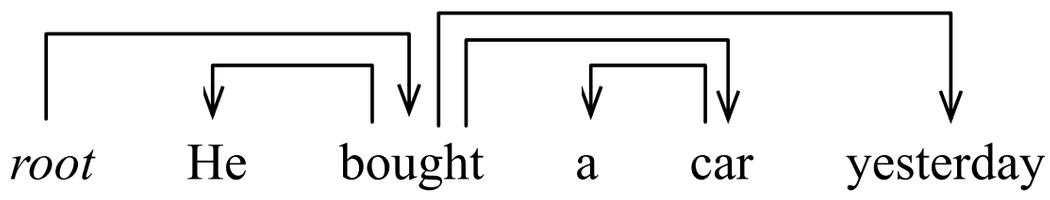
Phrase structures can be converted into dependency structures.

Apply head-finding rules recursively.

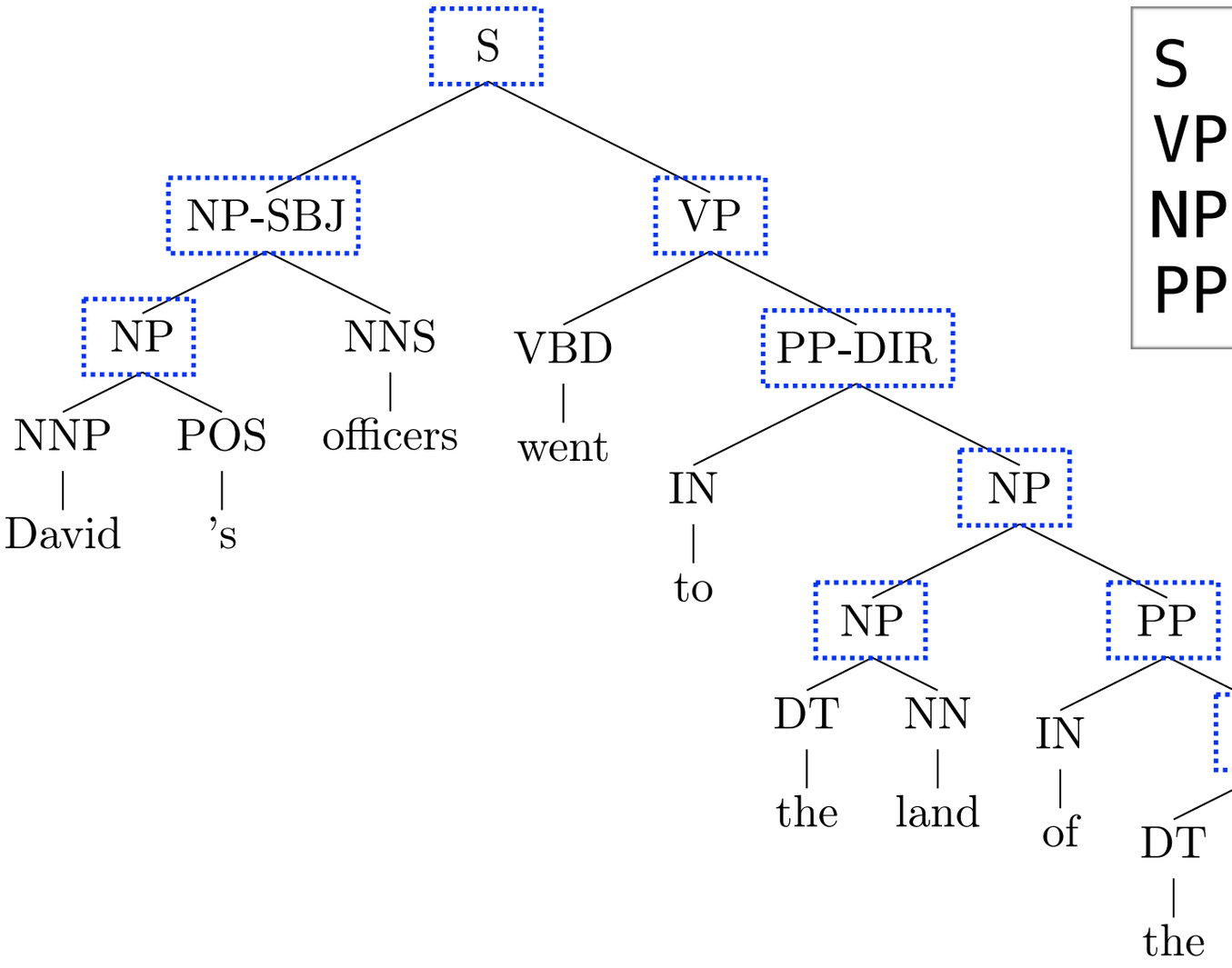
head-percolation rules, headrules



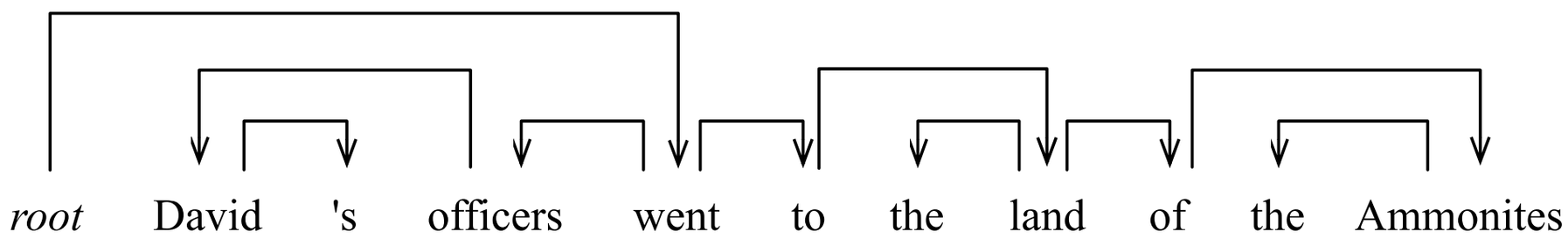



# Phrase To Dependency




```
S       r   VP
VP      l   VB*
NP      r   NN*;PRP;NP
PP      l   IN
```



# Attachment Scores

Assume each node has exactly one head except for the *root*.

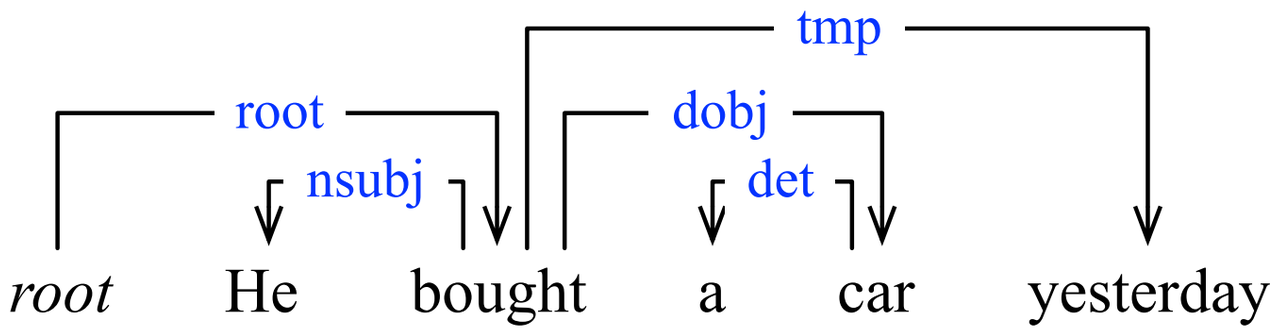
Unlabeled attachment score

How many nodes found correct heads.

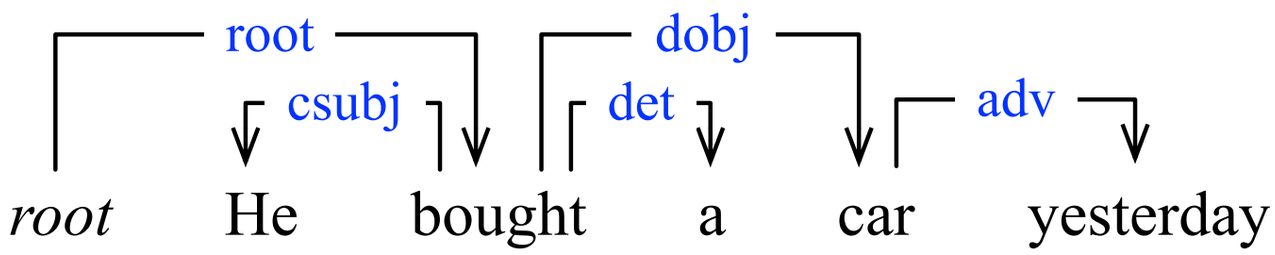
Labeled attachment score

How many nodes found correct heads and labels.

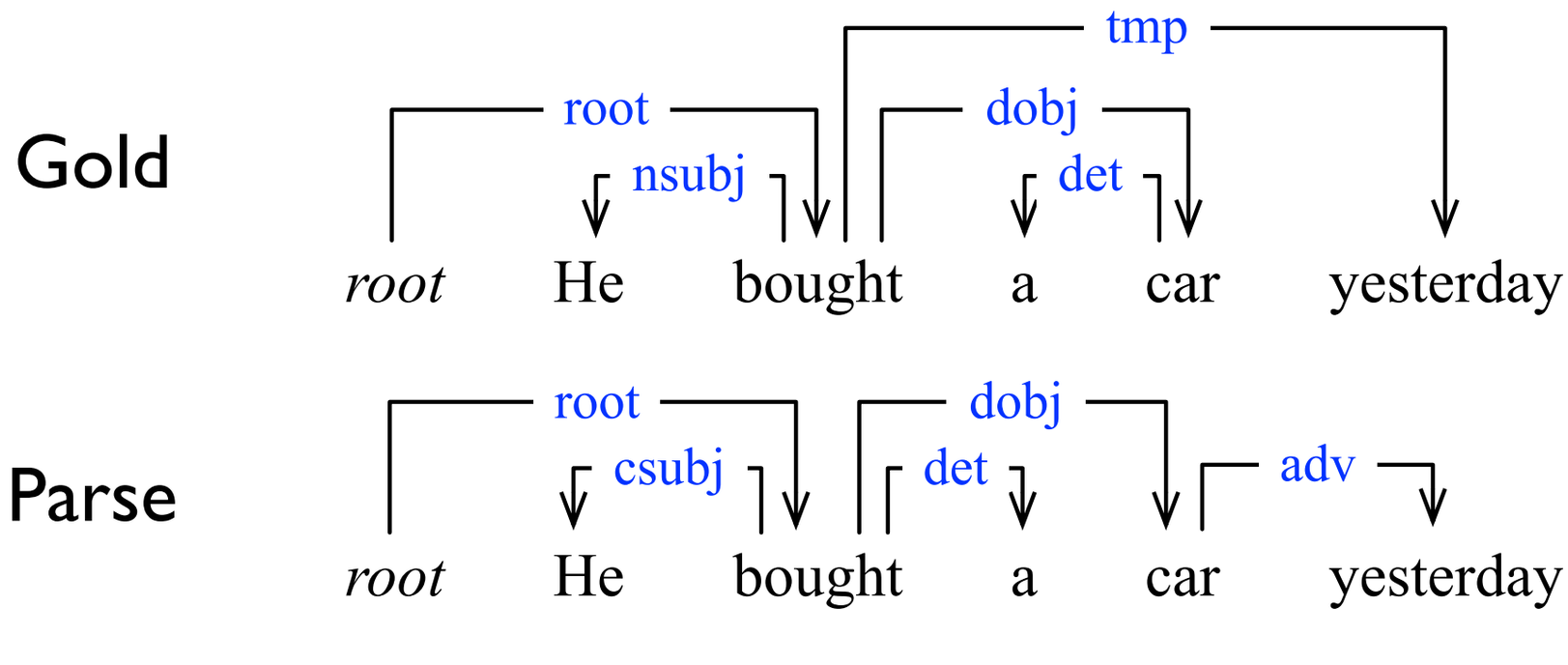
Gold




Parse



# Attachment Scores




**Unlabeled** attachment score

(He, bought)
(bought, *root*)
(a, bought) = 3/5
(car, bought)
(yesterday, car)

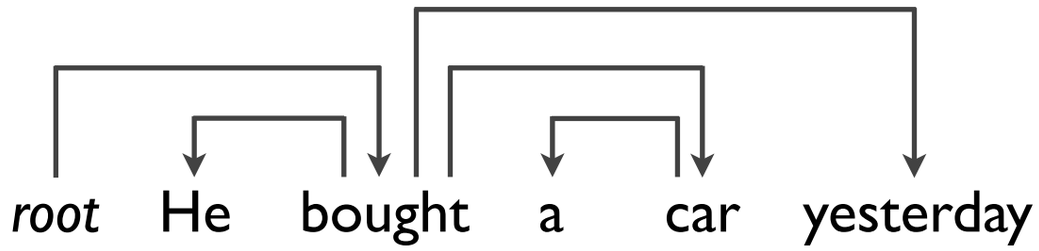
**Labeled** attachment score

(He, bought, csubj)
(bought, *root*, root)
(a, bought, det), = 2/5
(car, bought, dobj)
(yesterday, car, adv)

# Transition-based Parsing

- Nivre's arc-eager algorithm
  - Projective parsing algorithm with a worst-case complexity of $O(n)$.
  - S = stack, I = list of input tokens, A = set of arcs.

| | | |
|---|---|---|
| **Initialization** | $\langle S = [w_0], I = [w_1, \ldots, w_n], A = \emptyset \rangle$ | |
| **Termination** | $\langle S, [\,], A \rangle$ | |
| **Left-Arc** | $\langle w_i \vert S, w_j \vert I, A \rangle \Rightarrow \langle S, w_j \vert I, A \cup \{w_i \leftarrow w_j\} \rangle$ | $\neg\exists w_k.(w_i \leftarrow w_k) \in A$ |
| **Right-Arc** | $\langle w_i \vert S, w_j \vert I, A \rangle \Rightarrow \langle w_j \vert w_i \vert S, I, A \cup \{w_i \rightarrow w_j\} \rangle$ | $\neg\exists w_k.(w_k \rightarrow w_j) \in A$ |
| **Shift** | $\langle w_i \vert S, w_j \vert I, A \rangle \Rightarrow \langle w_j \vert w_i \vert S, I, A \rangle$ | |
| **Reduce** | $\langle w_i \vert S, w_j \vert I, A \rangle \Rightarrow \langle S, w_j \vert I, A \rangle$ | $\exists w_k.(w_i \leftarrow w_k) \in A$ |




| |
|---|
| |
| *root* |

| |
|---|
| He |
| bought |
| a |
| car |
| yesterday |

| |
|---|
| bought → yes.. |
| bought → car |
| a ← car |
| *root* → bought |
| He ← bought |

S

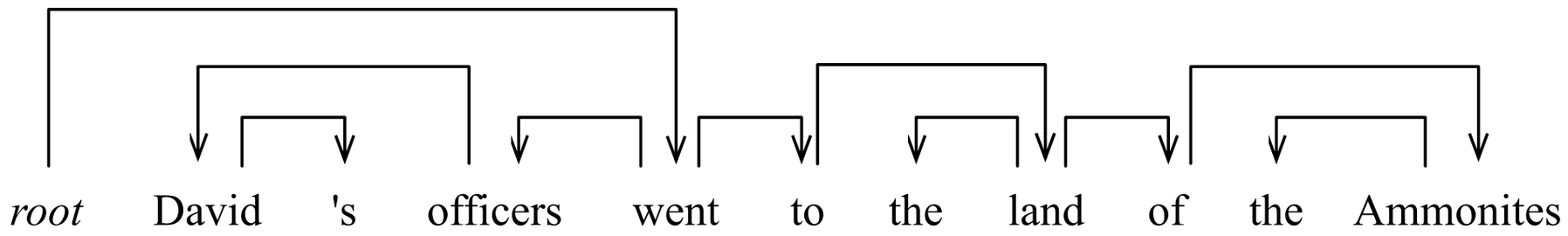
- Shift : ‘He’
- LeftArc : ‘He’ ← ‘bought’
- RightArc: *root* → ‘bought’
- Shift : ‘a’

I

- LeftArc : ‘a’ ← ‘car’
- RightArc: ‘bought’ → ‘car’
- Reduce: ‘car’
- RightArc: ‘bought’ → ‘yesterday’

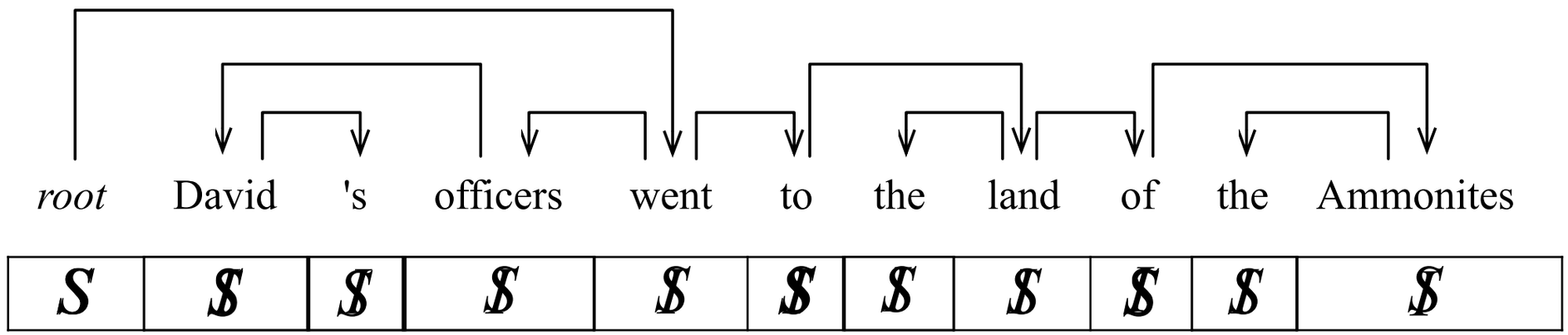
A

# Nivre's Arc-eager Algorithm




| | |
|---|---|
| **Initialization** | $\langle \mathbf{nil}, W, \emptyset \rangle$ |
| **Termination** | $\langle S, \mathbf{nil}, A \rangle$ |
| **Left-Reduce** | $\langle w_j w_i \vert S, I, A \rangle \rightarrow \langle w_j \vert S, I, A \cup \{(w_j, w_i)\} \rangle \quad \neg \exists w_k (w_k, w_i) \in A$ |
| **Right-Reduce** | $\langle w_j w_i \vert S, I, A \rangle \rightarrow \langle w_i \vert S, I, A \cup \{(w_i, w_j)\} \rangle \quad \neg \exists w_k (w_k, w_j) \in A$ |
| **Shift** | $\langle S, w_i \vert I, A \rangle \rightarrow \langle w_i \vert S, I, A \rangle$ |

# Nivre's Arc-eager Algorithm




- Initialize
- Shift: 'David'
- Right-Arc: David → 's
- Reduce: 's
- Left-Arc: David ← 'officers'
- Shift: officers
- Left-Arc: officers ← went
- Right-Arc: *root* → went
- Right-Arc: went → to
- Shift: the
- Left-Arc: the ← land
- Right-Arc: to → land
- Right-Arc: land → of
- Shift: 'the'
- Left-Arc: the ← Ammonites
- Right-Arc: of → Ammonites
- Terminate