Training Deterministic Parsers Using Non-Deterministic Oracles

Joakim Nivre

Uppsala University
Department of Linguistics and Philology
joakim.nivre@lingfil.uu.se

Joint work with Yoav Goldberg, Bar-Ilan University

Introduction

- ▶ Deterministic dependency parsing:
  - ▶ Very fast: 10^5 words per second
  - ▶ Fairly accurate: 2–3% below the state of the art
- ▶ How can we improve accuracy without losing speed?

Introduction

- ▶ Transition-based dependency parsing:
    - ▶ Define a transition system for dependency parsing
    - ▶ Train a classifier for predicting the next transition
    - ▶ Use the classifier to do deterministic parsing
- ▶ Current practice:
    - ▶ Train classifier on derivations produced by an oracle
    - ▶ Leads to error propagation at parsing time
    - ▶ Can be mitigated by using beam search – slowdown
- ▶ Novel idea:
    - ▶ Explore a larger search space during training
    - ▶ Allow the parser to make mistakes and recover
   - ▶ Requires a new type of oracle
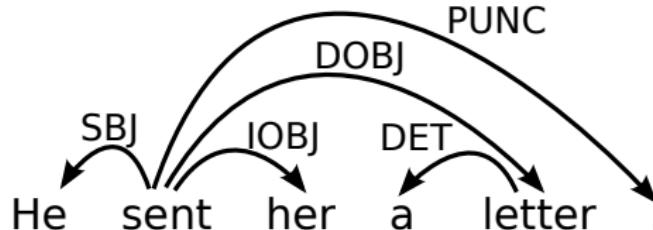
Outline

1. Transition-based dependency parsing
2. Old oracles – and why they are a problem
3. New oracles – and why they should help
4. Experiments

Dependency Trees

- A dependency tree is a labeled directed tree T with
    - a set V of nodes, labeled with words
    - a set A of arcs, labeled with dependency types
- Notation:
    - Arc (w_i, d, w_j) links head w_i to dependent w_j with label d
    - Shorthand: $w_i \xrightarrow{d} w_j \Leftrightarrow (w_i, d, w_j) \in A$

Transition System: Configurations

- ▸ A parser configuration is a triple $c = (S, B, A)$, where
    - ▸ S = a stack $[\ldots, w_i]_S$ of partially processed words,
    - ▸ B = a buffer $[w_j, \ldots]_B$ of remaining input word,
    - ▸ A = a set of labeled arcs (w_i, d, w_j).
- ▸ Initialization:

    $([\ ]_S, [w_1, \ldots, w_n]_B, \{\ \})$
- ▸ Termination:

    $(S, [\ ]_B, A)$

Transition System: Transitions

Left-Arc(d) $\dfrac{([\ldots, w_i]_S, \; [w_j, \ldots]_B, \; A)}{([\ldots]_S, \; [w_j, \ldots]_B, \; A \cup \{(w_j, d, w_i)\})}$ $\neg\textsc{head}(w_i)$

Right-Arc(d) $\dfrac{([\ldots, w_i]_S, \; [w_j, \ldots]_B, \; A)}{([\ldots, w_i, w_j]_S, \; [\ldots]_B, \; A \cup \{(w_i, d, w_j)\})}$

Reduce $\dfrac{([\ldots, w_i]_S, \; B, \; A)}{([\ldots]_S, \; B, \; A)}$ $\textsc{head}(w_i)$

Shift $\dfrac{([\ldots]_S, \; [w_i, \ldots]_B, \; A)}{([\ldots, w_i]_S, \; [\ldots]_B, \; A)}$

Parse Example

Transitions:

Stack	**Buffer**	**Arcs**
[ ]$_S$	[He, sent, her, a, letter, .]$_B$	

Parse Example

Transitions: SH

Stack

[He]$_S$
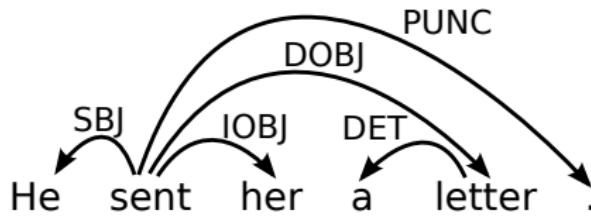
Buffer

[sent, her, a, letter, .]$_B$

Arcs

Parse Example

Transitions: SH-LA

Stack	**Buffer**	**Arcs**
[ ]$_S$	[sent, her, a, letter, .]$_B$	He $\overset{\text{SBJ}}{\longleftarrow}$ sent

Parse Example

Transitions: SH-LA-SH
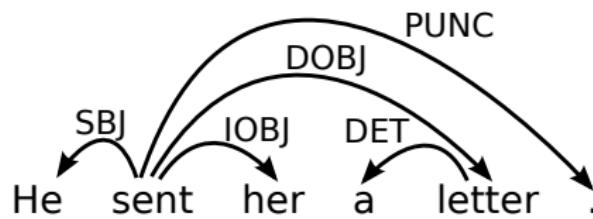
Stack	**Buffer**	**Arcs**
[sent]$_S$	[her, a, letter, .]$_B$	He $\overset{\text{SBJ}}{\longleftarrow}$ sent

Parse Example

Transitions: SH-LA-SH-RA

Stack

[sent, her]$_S$

Buffer

[a, letter, .]$_B$

Arcs

He $\overset{\text{SBJ}}{\longleftarrow}$ sent

sent $\overset{\text{IOBJ}}{\longrightarrow}$ her

He   sent   her   a   letter   .

Parse Example

Transitions: SH-LA-SH-RA-SH
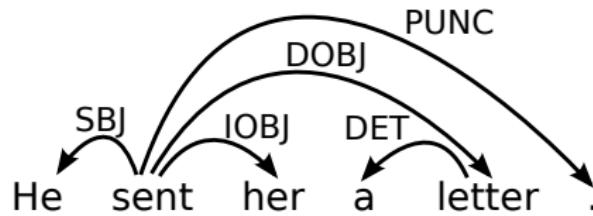
Stack

[sent, her, a]$_S$

Buffer

[letter, .]$_B$

Arcs

He $\overset{\text{SBJ}}{\longleftarrow}$ sent

sent $\overset{\text{IOBJ}}{\longrightarrow}$ her

Parse Example

Transitions: SH-LA-SH-RA-SH-LA

Stack
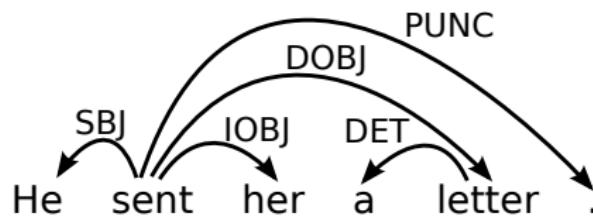
[sent, her]$_S$

Buffer

[letter, .]$_B$

Arcs

He $\overset{\text{SBJ}}{\longleftarrow}$ sent

sent $\overset{\text{IOBJ}}{\longrightarrow}$ her

a $\overset{\text{DET}}{\longleftarrow}$ letter

Parse Example

Transitions: SH-LA-SH-RA-SH-LA-RE

Stack
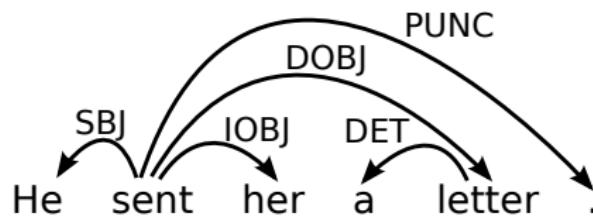
[sent]$_S$

Buffer

[letter, .]$_B$

Arcs

He $\overset{\text{SBJ}}{\longleftarrow}$ sent
sent $\overset{\text{IOBJ}}{\longrightarrow}$ her
a $\overset{\text{DET}}{\longleftarrow}$ letter

Parse Example

Transitions: SH-LA-SH-RA-SH-LA-RE-RA

Stack

[sent, letter]$_S$

Buffer

[.]$_B$
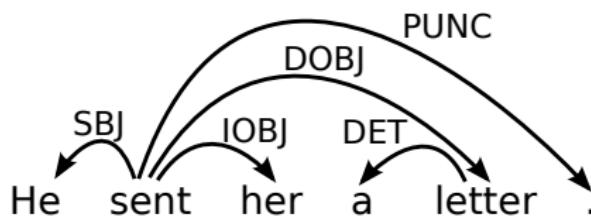
Arcs
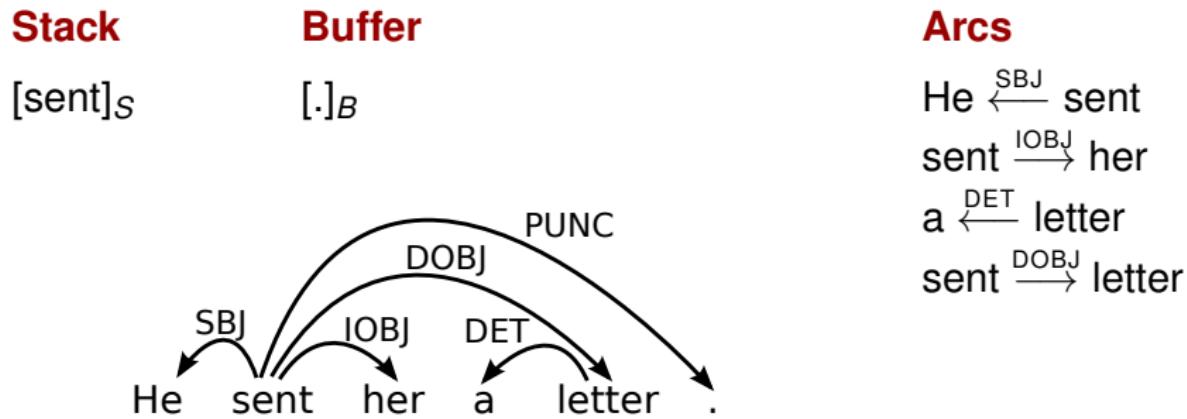
He $\overset{SBJ}{\longleftarrow}$ sent
sent $\overset{IOBJ}{\longrightarrow}$ her
a $\overset{DET}{\longleftarrow}$ letter
sent $\overset{DOBJ}{\longrightarrow}$ letter

Parse Example
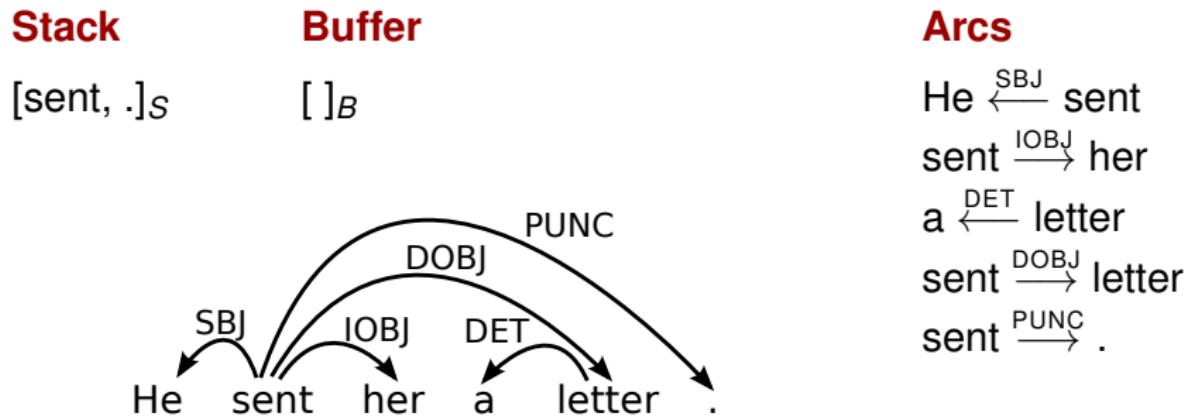
Transitions: SH-LA-SH-RA-SH-LA-RE-RA-RE

Stack

[sent]$_S$

Buffer

[.]$_B$

Arcs

He $\overset{\text{SBJ}}{\longleftarrow}$ sent
sent $\overset{\text{IOBJ}}{\longrightarrow}$ her
a $\overset{\text{DET}}{\longleftarrow}$ letter
sent $\overset{\text{DOBJ}}{\longrightarrow}$ letter

He   sent   her   a   letter   .

Parse Example

Transitions: SH-LA-SH-RA-SH-LA-RE-RA-RE-RA

Stack

[sent, .]$_S$

Buffer

[ ]$_B$

Arcs

He $\overset{\text{SBJ}}{\longleftarrow}$ sent

sent $\overset{\text{IOBJ}}{\longrightarrow}$ her

a $\overset{\text{DET}}{\longleftarrow}$ letter

sent $\overset{\text{DOBJ}}{\longrightarrow}$ letter

sent $\overset{\text{PUNC}}{\longrightarrow}$.

Classifier

▶ To guide the parser we use a (linear) classifier:

$$t^* = \underset{t}{\operatorname{argmax}} \, \mathbf{w} \cdot \mathbf{f}(c, t)$$

▶ History-based feature representation $\mathbf{f}(c, t)$:
  ▶ Features over input tokens relative to S and B
  ▶ Features over the (partial) dependency tree defined by A
  ▶ Features over the (partial) transition sequence

▶ Weight vector $\mathbf{w}$ learned from treebank data

Deterministic Parsing

$\text{PARSE}(w_1, \ldots, w_n, \mathbf{w})$
1 $c \leftarrow ([\,]_S, [w_1, \ldots, w_n]_B, \{\,\})$
2 **while** $B_c \neq [\,]$
3    $t^* \leftarrow \text{argmax}_t \, \mathbf{w} \cdot \mathbf{f}(c, t)$
4    $c \leftarrow t^*(c)$
5 **return** $T = (\{w_1, \ldots, w_n\}, A_c)$

Online Learning with an Oracle

```
LEARN({T_1, ..., T_N})
 1   w ← 0.0
 2   for i in 1..K
 3       for j in 1..N
 4           c ← ([ ]_S, [w_1, ..., w_{n_j}]_B, { })
 5           while B_c ≠ [ ]
 6               t* ← argmax_t w · f(c, t)
 7               t_o ← o(c, T_i)
 8               if t* ≠ t_o
 9                   w ← w + f(c, t_o) − f(c, t*)
10               c ← t_o(c)
11   return w
```

Online Learning with an Oracle

```
LEARN({T_1, ..., T_N})
 1   w ← 0.0
 2   for i in 1..K
 3       for j in 1..N
 4           c ← ([ ]_S, [w_1, ..., w_{n_j}]_B, { })
 5           while B_c ≠ [ ]
 6               t* ← argmax_t w · f(c, t)
 7               t_o ← o(c, T_i)
 8               if t* ≠ t_o
 9                   w ← w + f(c, t_o) − f(c, t*)
10               c ← t_o(c)
11   return w
```

▶ Oracle $o(c, T_i)$ returns the optimal transition for c and T_i

Standard Oracle for Arc-Eager Parsing

$$o(c, T) = \begin{cases} \text{Left-Arc} & \text{if } \text{top}(S_c) \leftarrow \text{first}(B_c) \text{ in } T \\ \text{Right-Arc} & \text{if } \text{top}(S_c) \rightarrow \text{first}(B_c) \text{ in } T \\ \text{Reduce} & \text{if } \exists w < \text{top}(S_c) : w \leftrightarrow \text{first}(B_c) \text{ in } T \\ \text{Shift} & \text{otherwise} \end{cases}$$

- ▶ Correct:
  - ▶ Derives T in a configuration sequence $C(o, T) = c_0, \ldots, c_m$
- ▶ Problems:
  - ▶ Deterministic: Ignores other derivations of T
  - ▶ Incomplete: Valid only for configurations in $C(o, T)$

Non-Determinism
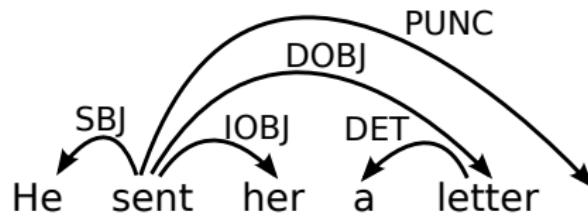
Transitions: SH-LA-SH-RA-SH-LA-RE-RA-RE-RA

Stack   **Buffer**   **Arcs**

[ ]$_S$ [He, sent, her, a, letter, .]$_B$

Non-Determinism

Transitions:　　　SH-LA-SH-RA-SH-LA-RE-RA-RE-RA

SH

Stack	**Buffer**	**Arcs**
[He]$_S$	[sent, her, a, letter, .]$_B$	
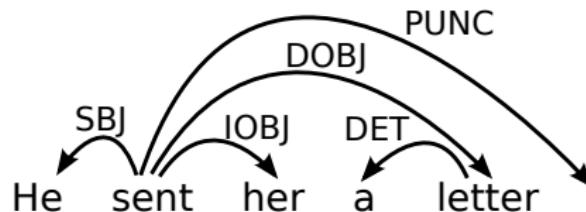
Non-Determinism

Transitions:	SH-LA-SH-RA-SH-LA-RE-RA-RE-RA
	SH-LA

Stack	**Buffer**	**Arcs**
[ ]$_S$	[sent, her, a, letter, .]$_B$	He $\xleftarrow{\text{SBJ}}$ sent
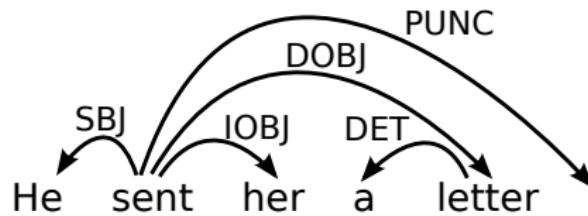
Non-Determinism

Transitions: SH-LA-SH-RA-SH-LA-RE-RA-RE-RA

SH-LA-SH

Stack	**Buffer**	**Arcs**
[sent]$_S$	[her, a, letter, .]$_B$	He $\xleftarrow{\text{SBJ}}$ sent

Non-Determinism

**Transitions:**

SH-LA-SH-RA-SH-LA-RE-RA-RE-RA

SH-LA-SH-RA

Stack
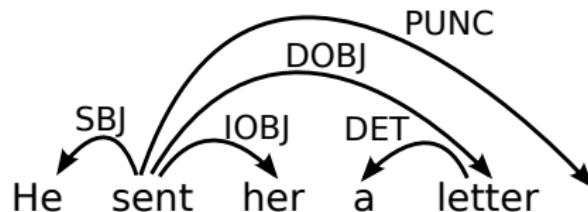
[sent, her]$_S$

Buffer

[a, letter, .]$_B$

Arcs

He $\overset{SBJ}{\longleftarrow}$ sent

sent $\overset{IOBJ}{\longrightarrow}$ her

Non-Determinism

Transitions:

SH-LA-SH-RA-SH-LA-RE-RA-RE-RA

SH-LA-SH-RA-RE

Stack
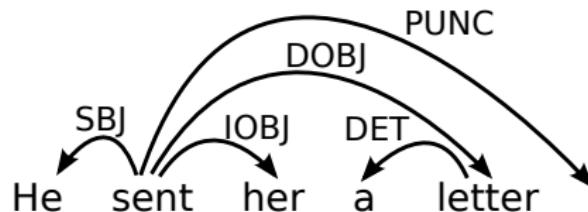
$[sent]_S$

Buffer

$[a, letter, .]_B$

Arcs

He $\overset{SBJ}{\longleftarrow}$ sent

sent $\overset{IOBJ}{\longrightarrow}$ her

Non-Determinism

Transitions:

SH-LA-SH-RA-SH-LA-RE-RA-RE-RA

SH-LA-SH-RA-RE-SH

Stack
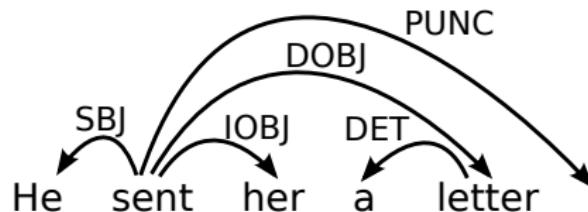
$[sent, a]_S$

Buffer

$[letter, .]_B$

Arcs

He $\overset{\text{SBJ}}{\longleftarrow}$ sent

sent $\overset{\text{IOBJ}}{\longrightarrow}$ her

Non-Determinism

Transitions:

SH-LA-SH-RA-SH-LA-RE-RA-RE-RA

SH-LA-SH-RA-RE-SH-LA

Stack
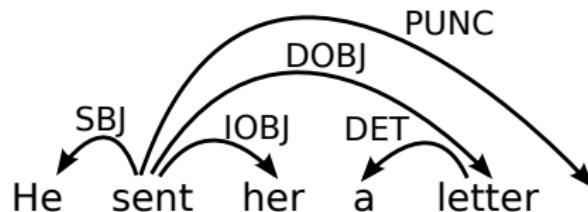
[sent]$_S$

Buffer

[letter, .]$_B$

Arcs

He $\xleftarrow{\text{SBJ}}$ sent
sent $\xrightarrow{\text{IOBJ}}$ her
a $\xleftarrow{\text{DET}}$ letter

Non-Determinism

Transitions:

SH-LA-SH-RA-SH-LA-RE-RA-RE-RA

SH-LA-SH-RA-RE-SH-LA-RA

Stack

[sent, letter]$_S$

Buffer

[.]$_B$
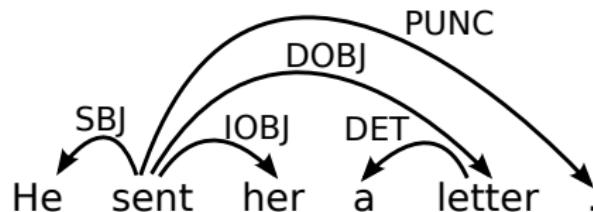
Arcs

He $\xleftarrow{SBJ}$ sent

sent $\xrightarrow{IOBJ}$ her

a $\xleftarrow{DET}$ letter

sent $\xrightarrow{DOBJ}$ letter

Non-Determinism

Transitions:

SH-LA-SH-RA-SH-LA-RE-RA-RE-RA

SH-LA-SH-RA-RE-SH-LA-RA-RE

Stack

$[\text{sent}]_S$

Buffer

$[.]_B$

Arcs

He $\overset{\text{SBJ}}{\longleftarrow}$ sent

sent $\overset{\text{IOBJ}}{\longrightarrow}$ her

a $\overset{\text{DET}}{\longleftarrow}$ letter

sent $\overset{\text{DOBJ}}{\longrightarrow}$ letter

PUNC
DOBJ
SBJ IOBJ DET
He sent her a letter .

Non-Determinism

Transitions:

SH-LA-SH-RA-SH-LA-RE-RA-RE-RA
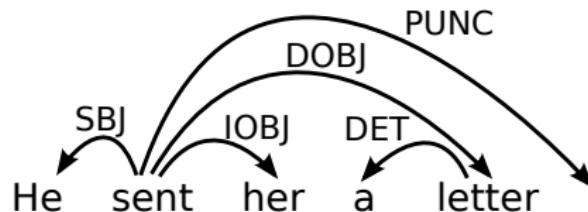
SH-LA-SH-RA-RE-SH-LA-RA-RE-RA

Stack

[sent, .]$_S$

Buffer

[ ]$_B$

Arcs

He $\xleftarrow{\text{SBJ}}$ sent

sent $\xrightarrow{\text{IOBJ}}$ her

a $\xleftarrow{\text{DET}}$ letter

sent $\xrightarrow{\text{DOBJ}}$ letter

sent $\xrightarrow{\text{PUNC}}$.

Non-Optimality

Transitions:

Stack	Buffer	Arcs
[ ]$_S$	[He, sent, her, a, letter, .]$_B$	
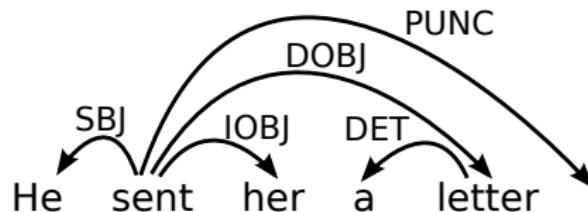
Non-Optimality

Transitions: SH

Stack   **Buffer**   **Arcs**
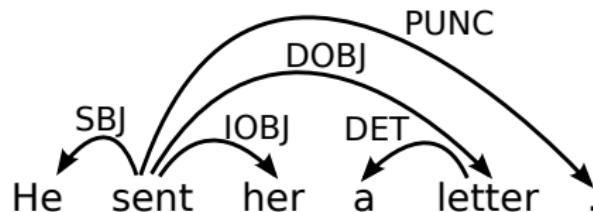
[He]$_S$ [sent, her, a, letter, .]$_B$

Non-Optimality

Transitions:

SH-LA

Stack

[ ]$_S$
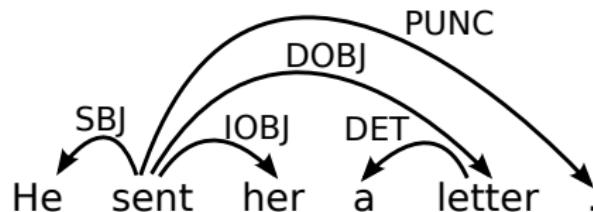
Buffer

[sent, her, a, letter, .]$_B$

Arcs

He $\xleftarrow{\text{SBJ}}$ sent

Non-Optimality

Transitions: SH-LA-SH
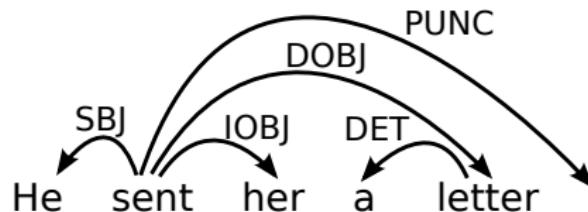
Stack

[sent]$_S$

Buffer

[her, a, letter, .]$_B$

Arcs

He $\xleftarrow{\text{SBJ}}$ sent

Non-Optimality

Transitions:
SH-LA-SH-SH

Stack
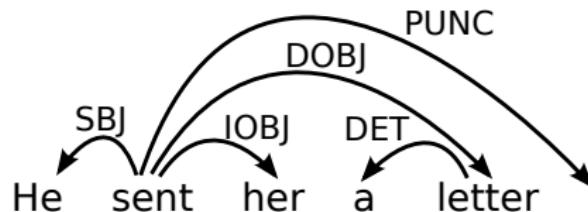
[sent, her]$_S$

Buffer

[a, letter, .]$_B$

Arcs

He $\xleftarrow{\text{SBJ}}$ sent

Non-Optimality

Transitions: SH-LA-SH-SH-SH

Stack
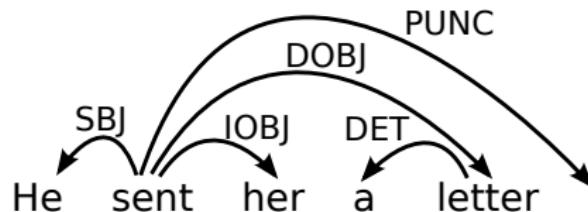
[sent, her, a]$_S$

Buffer

[letter, .]$_B$

Arcs

He $\xleftarrow{\text{SBJ}}$ sent

Non-Optimality

Transitions: SH-LA-SH-SH-SH-LA

Stack
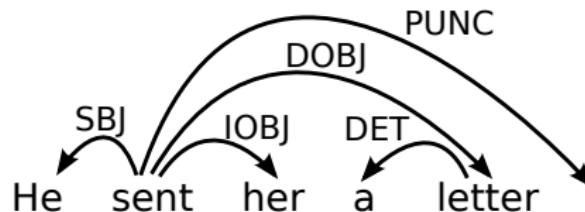
[sent, her]$_S$

Buffer

[letter, .]$_B$

Arcs

He $\xleftarrow{\text{SBJ}}$ sent
a $\xleftarrow{\text{DET}}$ letter

Non-Optimality

Transitions:

SH-LA-SH-SH-SH-LA-SH

Stack
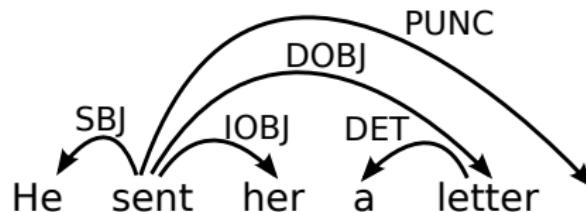
[sent, her, letter]$_S$

Buffer

[.]$_B$

Arcs

He $\xleftarrow{\text{SBJ}}$ sent
a $\xleftarrow{\text{DET}}$ letter

He sent her a letter .

SBJ, IOBJ, DET, DOBJ, PUNC

Non-Optimality

Transitions: SH-LA-SH-SH-SH-LA-SH-SH [2/5]

Stack

[sent, her, letter, .]$_S$

Buffer

[ ]$_B$

Arcs

He $\xleftarrow{\text{SBJ}}$ sent

a $\xleftarrow{\text{DET}}$ letter

Non-Optimality

Transitions: SH-LA-SH-SH-SH-LA-SH-SH [2/5]

Stack	**Buffer**	**Arcs**
[ ]$_S$	[He, sent, her, a, letter, .]$_B$	
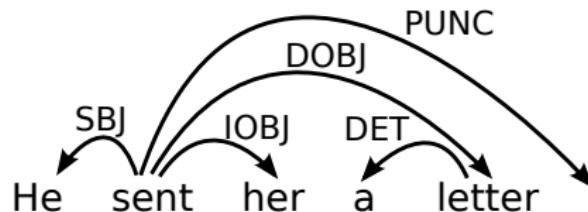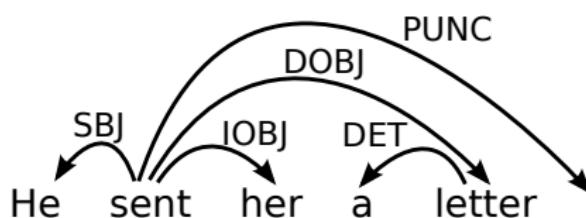
Non-Optimality

Transitions: SH-LA-SH-SH-SH-LA-SH-SH   [2/5]

SH

Stack   **Buffer**   **Arcs**
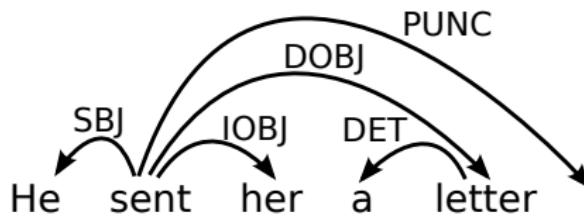
[He]$_S$ [sent, her, a, letter, .]$_B$

Non-Optimality

Transitions: SH-LA-SH-SH-SH-LA-SH-SH [2/5]

SH-LA

Stack   **Buffer**   **Arcs**
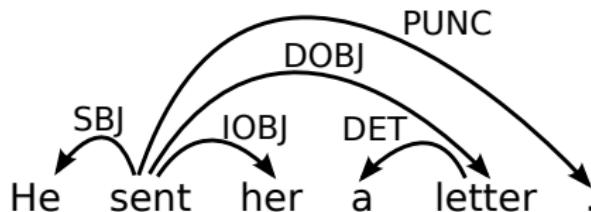
[ ]$_S$ [sent, her, a, letter, .]$_B$ He $\xleftarrow{\text{SBJ}}$ sent

Non-Optimality

**Transitions:**   SH-LA-SH-SH-SH-LA-SH-SH [2/5]

SH-LA-SH

Stack   **Buffer**   **Arcs**

[sent]$_S$ [her, a, letter, .]$_B$ He $\xleftarrow{SBJ}$ sent

Non-Optimality

Transitions: SH-LA-SH-SH-SH-LA-SH-SH [2/5]

SH-LA-SH-SH

Stack	**Buffer**	**Arcs**
[sent, her]$_S$	[a, letter, .]$_B$	He $\xleftarrow{\text{SBJ}}$ sent
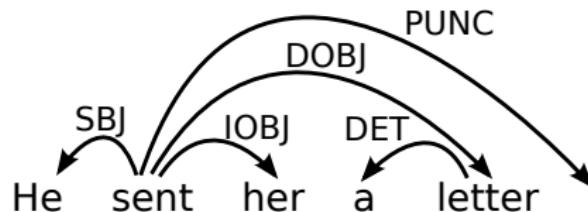
Non-Optimality

Transitions: SH-LA-SH-SH-SH-LA-SH-SH   [2/5]

SH-LA-SH-SH-SH

Stack
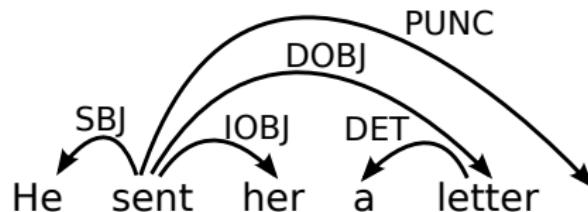
[sent, her, a]$_S$

Buffer

[letter, .]$_B$

Arcs

He $\xleftarrow{\text{SBJ}}$ sent

Non-Optimality

Transitions:

SH-LA-SH-SH-SH-LA-SH-SH [2/5]

SH-LA-SH-SH-SH-LA

Stack
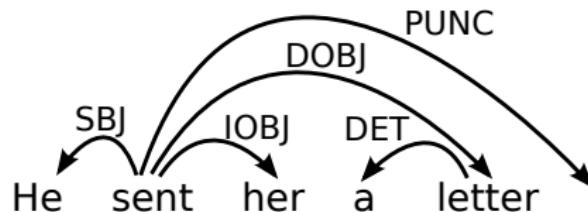
[sent, her]$_S$

Buffer

[letter, .]$_B$

Arcs

He $\xleftarrow{\text{SBJ}}$ sent

a $\xleftarrow{\text{DET}}$ letter

Non-Optimality

Transitions:

SH-LA-SH-SH-SH-LA-SH-SH [2/5]

SH-LA-SH-SH-SH-LA-LA

Stack
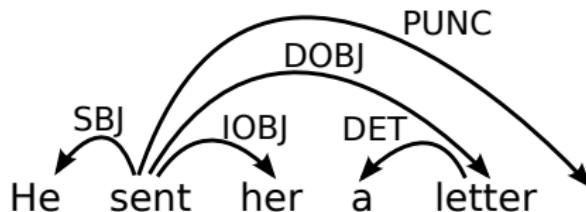
$[sent]_S$

Buffer

$[letter, .]_B$

Arcs

He $\xleftarrow{SBJ}$ sent
a $\xleftarrow{DET}$ letter
her $\xleftarrow{X}$ letter

Non-Optimality

Transitions:

SH-LA-SH-SH-SH-LA-SH-SH [2/5]
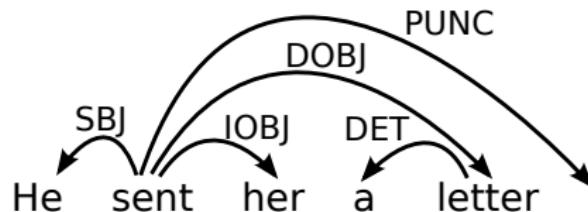
SH-LA-SH-SH-SH-LA-LA-RA

Stack

[sent, letter]$_S$

Buffer

[.]$_B$

Arcs

He $\overset{\text{SBJ}}{\longleftarrow}$ sent

a $\overset{\text{DET}}{\longleftarrow}$ letter

her $\overset{\text{X}}{\longleftarrow}$ letter

sent $\overset{\text{DOBJ}}{\longrightarrow}$ letter

Non-Optimality

Transitions:

SH-LA-SH-SH-SH-LA-SH-SH   [2/5]
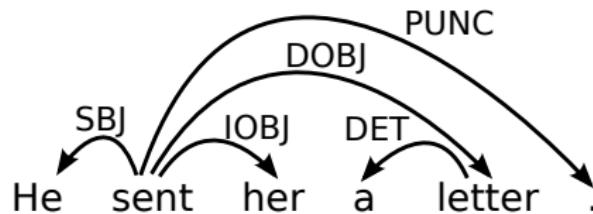
SH-LA-SH-SH-SH-LA-LA-RA-RE

Stack

[sent]$_S$

Buffer

[.]$_B$

Arcs

He $\overset{SBJ}{\longleftarrow}$ sent

a $\overset{DET}{\longleftarrow}$ letter

her $\overset{X}{\longleftarrow}$ letter

sent $\overset{DOBJ}{\longrightarrow}$ letter

Non-Optimality

Transitions:

SH-LA-SH-SH-SH-LA-SH-SH   [2/5]
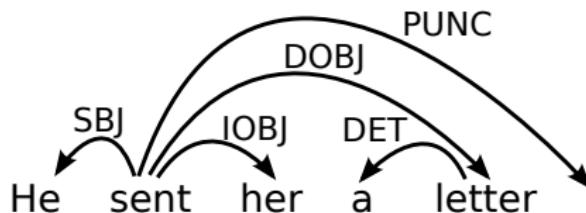
SH-LA-SH-SH-SH-LA-LA-RA-RE-RA   [4/5]

Stack

[sent, .]$_S$

Buffer

[ ]$_B$

Arcs

He $\overset{\text{SBJ}}{\longleftarrow}$ sent

a $\overset{\text{DET}}{\longleftarrow}$ letter

her $\overset{\text{X}}{\longleftarrow}$ letter

sent $\overset{\text{DOBJ}}{\longrightarrow}$ letter

sent $\overset{\text{PUNC}}{\longrightarrow}$.

Rethinking Oracles

- ▶ New idea:
  - ▶ A transition is optimal if the best tree remains reachable
  - ▶ Best tree = $\text{argmin}_{T'}\, \mathcal{L}(T, T')$
- ▶ New view of oracle:
  - ▶ Boolean function $o(c, t, T) = $ **true** if t is optimal for c and T
  - ▶ Non-deterministic: More than one transition can be optimal
  - ▶ Complete: Correct for all configurations
- ▶ New problem:
  - ▶ How do we know which trees are reachable?

Reachability for Arcs and Trees

- ▶ Arc reachability:
  - ▶ An arc $w_i \rightarrow w_j$ is reachable in c iff $w_i \rightarrow w_j \in A_c$, or $w_i \in S_c \cup B_c$ and $w_j \in B_c$ (same for $w_i \leftarrow w_j$)
- ▶ Tree reachability:
  - ▶ A (projective) tree T is reachable in c iff every arc in T is reachable in c
- ▶ Arc-decomposable system:
  - ▶ Tree reachability reduces to arc reachability
  - ▶ Holds for some transition systems but not all

A New Oracle

$$\mathcal{R}(c) \equiv \{a \,|\, a \text{ is an arc reachable in } c\}$$

$$o(c, t, T) = \begin{cases} \textbf{true} & \text{if } [\mathcal{R}(c) - \mathcal{R}(t(c))] \cap T = \emptyset \\ \textbf{false} & \text{otherwise} \end{cases}$$

Case by Case

► Notation: s = word on top of stack, b = first word in buffer

$$o(c, \text{LA}, T) = \begin{cases} \textbf{false} & \text{if } \exists w \in B_c : s \leftrightarrow w \in T \text{ (except } s \leftarrow b) \\ \textbf{true} & \text{otherwise} \end{cases}$$

$$o(c, \text{RA}, T) = \begin{cases} \textbf{false} & \text{if } \exists w \in S_c : w \leftrightarrow b \in T \text{ (except } s \rightarrow b) \\ \textbf{true} & \text{otherwise} \end{cases}$$

$$o(c, \text{RE}, T) = \begin{cases} \textbf{false} & \text{if } \exists w \in B_c : s \rightarrow w \in T \\ \textbf{true} & \text{otherwise} \end{cases}$$

$$o(c, \text{SH}, T) = \begin{cases} \textbf{false} & \text{if } \exists w \in S_c : w \leftrightarrow b \in T \\ \textbf{true} & \text{otherwise} \end{cases}$$

A New Learning Algorithm

```
LEARN({T₁, ..., Tₙ})
1    w ← 0.0
2    for i in 1..K
3        for j in 1..N
4            c ← ([ ]ₛ, [w₁, ..., wₙⱼ]ᵦ, { })
5            while Bc ≠ [ ]
6                t* ← argmaxₜ w · f(c, t)
7                tₒ ← argmaxₜ∈{t|o(c,t,Tᵢ)} w · f(c, t)
8                if t* ≠ tₒ
9                    w ← w + f(c, tₒ) − f(c, t*)
10               c ← CHOICE(tₒ(c), t*(c))
11   return w
```

A New Learning Algorithm
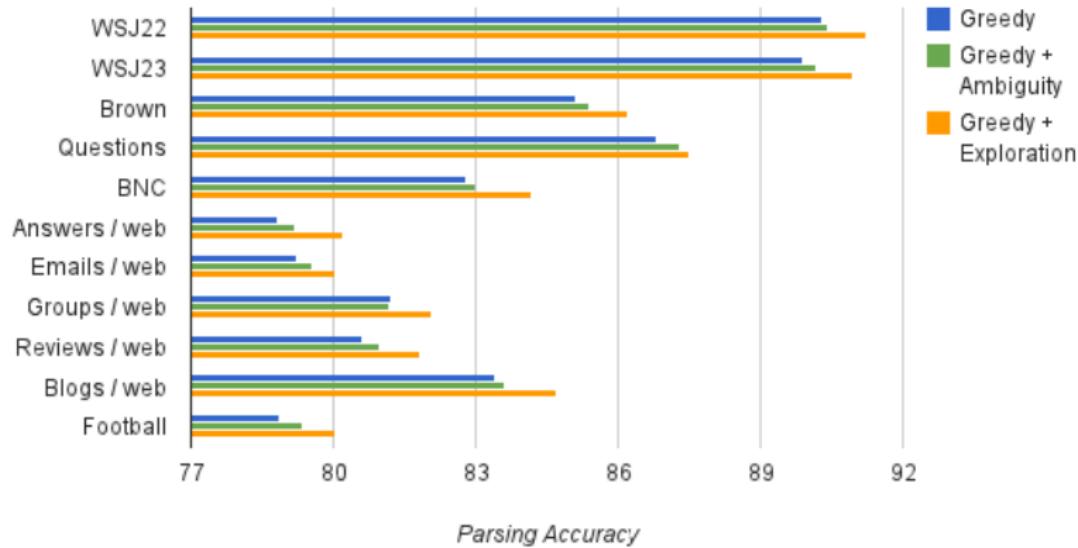
$\text{LEARN}(\{T_1, \ldots, T_N\})$
1 $\mathbf{w} \leftarrow 0.0$
2 **for** i in $1..K$
3   **for** j in $1..N$
4      $c \leftarrow ([\,]_S, [w_1, \ldots, w_{n_j}]_B, \{\,\})$
5   **while** $B_c \neq [\,]$
6        $t^* \leftarrow \text{argmax}_t \, \mathbf{w} \cdot \mathbf{f}(c, t)$
7        $t_o \leftarrow \text{argmax}_{t \in \{t | o(c, t, T_i)\}} \, \mathbf{w} \cdot \mathbf{f}(c, t)$
8   **if** $t^* \neq t_o$
9          $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(c, t_o) - \mathbf{f}(c, t^*)$
10        $c \leftarrow \text{CHOICE}(t_o(c), t^*(c))$
11 **return w**
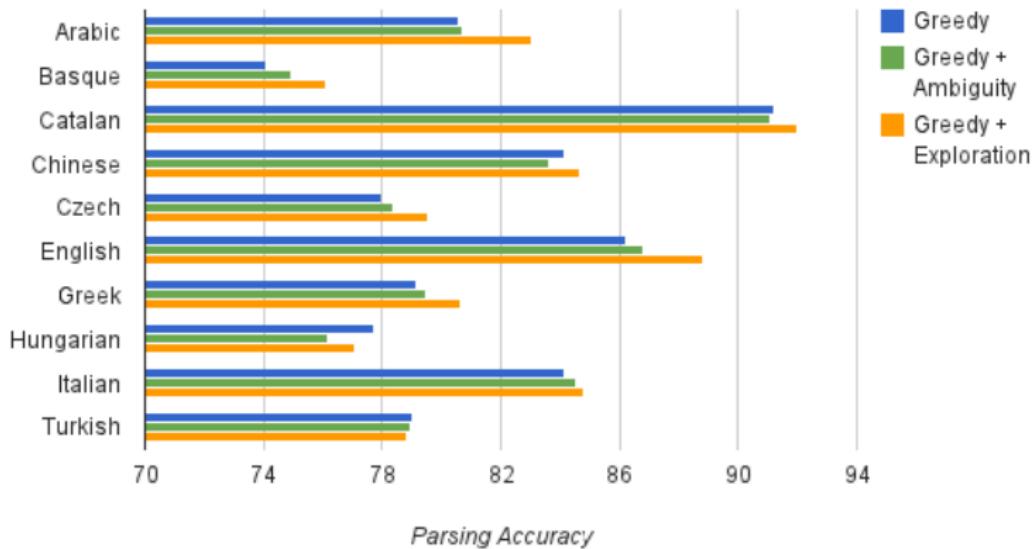
Experimental Evaluation

- ► Data sets:
  - ► English treebanks: WSJ, Brown, BNC, Google Web
  - ► Multilingual: CoNLL 2007 Shared Task
- ► Settings:
  - ► Greedy: Old learning algorithm
  - ► Greedy + Ambiguity: CHOICE$(t_o(c), t^*(c)) = t_o(c)$
 - ► Greedy + Exploration: Random CHOICE$(t_o(c), t^*(c))$

English Results

Multilingual Results

Conclusion

- Exploring a larger search space at training time helps
  - Allowing non-canonical derivations (spurious ambiguity)
  - Learning optimal transitions in non-optimal configurations
- Requires a new type of oracle
  - Non-deterministic: more than one transition may be optimal
  - Complete: optimality defined for all configurations
- Parsing remains deterministic (fast)