L95: Natural Language Syntax and Parsing
3) Dependency Parsing

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge

Reminder:

So far we have:

- Used CFGs to define a set of strings through legal structures
- Used the CKY algorithm to find all structures for a given string
- Used PCFGs to assign probabilities to the structures deriving the string
- Used a modified CKY algorithm to find the best structure from all the possibilities

Reminder:

So far we have: (for statistical parsing more generally we need...)

- Used CFGs to define a set of strings through legal structures (a grammar)
- Used the CKY algorithm to find all structures for a given string (a parsing algorithm)
- Used PCFGs to assign probabilities to the structures deriving the string (a scoring model for parses)
- Used a modified CKY algorithm to find the best structure from all the possibilities (an algorithm for finding best parse)

Reminder:

Recall that:

$$\hat{T}(W) = \underset{\textit{trees that yield } W}{\operatorname{argmax}} P(T|W)$$

- finding Ts that yield W requires a parsing algorithm over a grammar
- knowing $P(T|W)$ requires a probabilistic model over T
- argmax requires an algorithm for finding best parse

More generally:

$$\hat{T}(W) = \underset{\textit{trees that yield } W}{\operatorname{argmax}} Score(T|W)$$

- Generative models use the product of estimated probabilities of parse pieces to find $P(T|W)$ but we could use other scoring functions...

Reminder:

Recall that:

$$\hat{T}(W) = \underset{\textit{trees that yield } W}{\operatorname{argmax}} P(T|W)$$

- finding Ts that yield W requires a parsing algorithm over a grammar
- knowing $P(T|W)$ requires a probabilistic model over T
- argmax requires an algorithm for finding best parse

More generally:

$$\hat{T}(W) = \underset{\textit{trees that yield } W}{\operatorname{argmax}} Score(T|W)$$

- Generative models use the product of estimated probabilities of parse pieces to find $P(T|W)$ but we could use other scoring functions...

Generative models have some issues

Recall that:

$$P(T,W) = \prod_{i=1}^{n} P(A_i \rightarrow B_i)$$

- But $P(T,W) = P(T)P(W|T)$ and that $P(W|T) = 1$ so
  $P(T,W) = P(T)$ and thus $P(T) = \prod_{i=1}^{n} P(A_i \rightarrow B_i)$
- How do we know how to break the parse into pieces?
- Is the independence assumption valid?
- Generative models simultaneously model the tree and the string—**discriminative models** define $P(T|W)$ directly
- Models are **discriminative** because they compare the correct parse against incorrect parses in training to set parameters... can use machine learning approaches.
- First we're going to learn about a grammar that we will use to define T within a discriminative model—**dependency grammars**.

Generative models have some issues

Recall that:

$$P(T, W) = \prod_{i=1}^{n} P(A_i \rightarrow B_i)$$

- But $P(T, W) = P(T)P(W|T)$ and that $P(W|T) = 1$ so

  $P(T, W) = P(T)$ and thus $P(T) = \prod_{i=1}^{n} P(A_i \rightarrow B_i)$
- How do we know how to break the parse into pieces?
- Is the independence assumption valid?
- Generative models simultaneously model the tree and the string—**discriminative models** define $P(T|W)$ directly
- Models are **discriminative** because they compare the correct parse against incorrect parses in training to set parameters... can use machine learning approaches.
- First we're going to learn about a grammar that we will use to define T within a discriminative model—**dependency grammars**.

Generative models have some issues

Recall that:

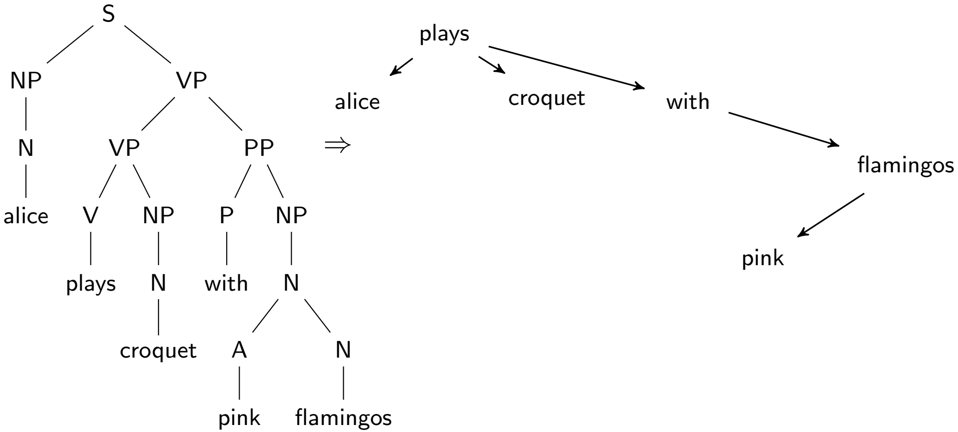
$$P(T,W) = \prod_{i=1}^{n} P(A_i \rightarrow B_i)$$

- But $P(T,W) = P(T)P(W|T)$ and that $P(W|T) = 1$ so

  $P(T,W) = P(T)$ and thus $P(T) = \prod_{i=1}^{n} P(A_i \rightarrow B_i)$
- How do we know how to break the parse into pieces?
- Is the independence assumption valid?
- Generative models simultaneously model the tree and the string—**discriminative models** define $P(T|W)$ directly
- Models are **discriminative** because they compare the correct parse against incorrect parses in training to set parameters... can use machine learning approaches.
- First we're going to learn about a grammar that we will use to define T within a discriminative model—**dependency grammars**.

A **dependency tree** is a directed graph

A **dependency tree** is a directed graph representation of a string—each edge represents a grammatical relationship between the symbols.



A **dependency grammar** deriving **dependency trees**

Formally $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$ where:

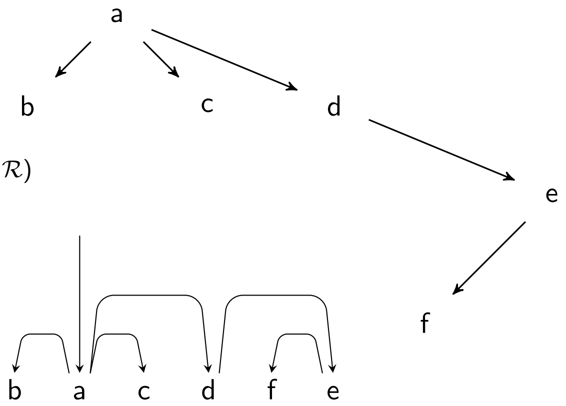
- Σ is the finite set of alphabet symbols
- $\mathcal{D}$ is the set of symbols to indicate whether the dependent symbol (the one on the RHS of the rule) will be located on the left or right of the current item within the string $\mathcal{D} = \{\mathcal{L}, \mathcal{R}\}$
- s is the root symbol for the dependency tree (we will use $s \in \Sigma$ but sometimes a special extra symbol is used)
- $\perp$ is a symbol to indicate an allowed endpoint to the graph
- $\mathcal{P}$ is a set of rules for generating dependencies:
  $\mathcal{P} = \{(\alpha \to \beta,\ d) \mid \alpha \in (\Sigma\ \cup\ s),\ \beta \in (\Sigma \cup \perp),\ d \in \mathcal{D}\}$

In dependency grammars we refer to the term on the LHS of a rule as the **head** and the RHS as the **dependent** (as opposed to *parents* and *children* in phrase structure grammars).

Dependency trees have **several representations**

Two diagrammatic representations of a dependency tree for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$ where:

$$
\begin{aligned}
\Sigma &= \{a \ldots f\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \text{a} \\
\mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
&\quad (d \rightarrow e, \mathcal{R}) \\
&\quad (e \rightarrow f, \mathcal{L}) \\
&\quad (b \rightarrow \perp, \mathcal{L} \mid \perp, \mathcal{R}) \\
&\quad (c \rightarrow \perp, \mathcal{L} \mid \perp, \mathcal{R}) \\
&\quad (f \rightarrow \perp, \mathcal{L} \mid \perp, \mathcal{R})\}
\end{aligned}
$$



The same rules would have been used to generate the string *badfec*.

Useful when there is flexibility in the symbol order of grammatical strings.

Recall: shift-reduce parsers and **deterministic** languages

LR(k) Shift-reduce parsers are most useful for recognising the strings of **deterministic** languages (languages where no string has more than one analysis) which have been described by an unambiguous grammar.

Quick reminder:

- The parsing algorithm has two actions: SHIFT and REDUCE
- Initially the input string is held in the buffer and the stack is empty.
- Symbols are **shifted** from the buffer to the stack
- When the top items of the stack match the RHS of a rule in the grammar then they are **reduced**, that is, they are replaced with the LHS of that rule.
- k refers to the look-ahead.

Shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$
\begin{array}{rcl}
\Sigma & = & \{a, b, c\} \\
\mathcal{N} & = & \{S, A, B, C, D\} \\
s & = & \mathrm{S} \\
\mathcal{P} & = & \{S \rightarrow A\ B, \\
 & & A \rightarrow a, \\
 & & B \rightarrow b\ C, \\
 & & C \rightarrow c\ D, \\
 & & D \rightarrow d\}
\end{array}
$$

STACK	BUFFER	ACTION
	abcd	

Shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$
\begin{array}{rcl}
\Sigma & = & \{a, b, c\} \\
\mathcal{N} & = & \{S, A, B, C, D\} \\
s & = & \mathrm{S} \\
\mathcal{P} & = & \{S \rightarrow A\ B, \\
 & & A \rightarrow a, \\
 & & B \rightarrow b\ C, \\
 & & C \rightarrow c\ D, \\
 & & D \rightarrow d\}
\end{array}
$$

STACK	BUFFER	ACTION
	abcd	SHIFT

Shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$
\begin{aligned}
\Sigma &= \{a, b, c\} \\
\mathcal{N} &= \{S, A, B, C, D\} \\
s &= \mathrm{S} \\
\mathcal{P} &= \{S \rightarrow A\ B, \\
&\quad A \rightarrow a, \\
&\quad B \rightarrow b\ C, \\
&\quad C \rightarrow c\ D, \\
&\quad D \rightarrow d\}
\end{aligned}
$$

STACK	BUFFER	ACTION
	abcd	SHIFT
a	bcd	

Shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$
\begin{aligned}
\Sigma &= \{a, b, c\} \\
\mathcal{N} &= \{S, A, B, C, D\} \\
s &= \mathrm{S} \\
\mathcal{P} &= \{S \rightarrow A\ B, \\
&\quad A \rightarrow a, \\
&\quad B \rightarrow b\ C, \\
&\quad C \rightarrow c\ D, \\
&\quad D \rightarrow d\}
\end{aligned}
$$

STACK	BUFFER	ACTION
	abcd	SHIFT
a	bcd	REDUCE

Shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$
\begin{aligned}
\Sigma &= \{a, b, c\} \\
\mathcal{N} &= \{S, A, B, C, D\} \\
s &= \mathrm{S} \\
\mathcal{P} &= \{S \rightarrow A\,B, \\
&\quad A \rightarrow a, \\
&\quad B \rightarrow b\,C, \\
&\quad C \rightarrow c\,D, \\
&\quad D \rightarrow d\}
\end{aligned}
$$

STACK	BUFFER	ACTION
	abcd	SHIFT
a	bcd	REDUCE
A	bcd	

Shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$
\begin{aligned}
\Sigma &= \{a, b, c\} \\
\mathcal{N} &= \{S, A, B, C, D\} \\
s &= \mathrm{S} \\
\mathcal{P} &= \{S \rightarrow A\ B, \\
&\quad A \rightarrow a, \\
&\quad B \rightarrow b\ C, \\
&\quad C \rightarrow c\ D, \\
&\quad D \rightarrow d\}
\end{aligned}
$$

STACK	BUFFER	ACTION
	abcd	SHIFT
a	bcd	REDUCE
A	bcd	SHIFT

Shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$
\begin{aligned}
\Sigma &= \{a, b, c\} \\
\mathcal{N} &= \{S, A, B, C, D\} \\
s &= \mathrm{S} \\
\mathcal{P} &= \{S \rightarrow A\,B, \\
& \quad A \rightarrow a, \\
& \quad B \rightarrow b\,C, \\
& \quad C \rightarrow c\,D, \\
& \quad D \rightarrow d\}
\end{aligned}
$$

STACK	BUFFER	ACTION
	abcd	SHIFT
a	bcd	REDUCE
A	bcd	SHIFT
Ab	cd	

Shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$
\begin{aligned}
\Sigma &= \{a, b, c\} \\
\mathcal{N} &= \{S, A, B, C, D\} \\
s &= \mathrm{S} \\
\mathcal{P} &= \{S \rightarrow A\,B, \\
&\quad A \rightarrow a, \\
&\quad B \rightarrow b\,C, \\
&\quad C \rightarrow c\,D, \\
&\quad D \rightarrow d\}
\end{aligned}
$$

STACK	BUFFER	ACTION
	abcd	SHIFT
a	bcd	REDUCE
A	bcd	SHIFT
Ab	cd	SHIFT

Shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$
\begin{aligned}
\Sigma &= \{a, b, c\} \\
\mathcal{N} &= \{S, A, B, C, D\} \\
s &= \mathsf{S} \\
\mathcal{P} &= \{S \rightarrow A\,B, \\
& \quad A \rightarrow a, \\
& \quad B \rightarrow b\,C, \\
& \quad C \rightarrow c\,D, \\
& \quad D \rightarrow d\}
\end{aligned}
$$

STACK	BUFFER	ACTION
	abcd	SHIFT
a	bcd	REDUCE
A	bcd	SHIFT
Ab	cd	SHIFT
Abc	d	

Shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$
\begin{aligned}
\Sigma &= \{a, b, c\} \\
\mathcal{N} &= \{S, A, B, C, D\} \\
s &= \mathrm{S} \\
\mathcal{P} &= \{S \rightarrow A\,B, \\
&\quad A \rightarrow a, \\
&\quad B \rightarrow b\ C, \\
&\quad C \rightarrow c\ D, \\
&\quad D \rightarrow d\}
\end{aligned}
$$

STACK	BUFFER	ACTION
	abcd	SHIFT
a	bcd	REDUCE
A	bcd	SHIFT
Ab	cd	SHIFT
Abc	d	SHIFT

Shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$
\begin{aligned}
\Sigma &= \{a, b, c\} \\
\mathcal{N} &= \{S, A, B, C, D\} \\
s &= \mathrm{S} \\
\mathcal{P} &= \{S \rightarrow A\ B, \\
&\quad A \rightarrow a, \\
&\quad B \rightarrow b\ C, \\
&\quad C \rightarrow c\ D, \\
&\quad D \rightarrow d\}
\end{aligned}
$$

STACK	BUFFER	ACTION
	abcd	SHIFT
a	bcd	REDUCE
A	bcd	SHIFT
Ab	cd	SHIFT
Abc	d	SHIFT
Abcd		

Shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$
\begin{aligned}
\Sigma &= \{a, b, c\} \\
\mathcal{N} &= \{S, A, B, C, D\} \\
s &= \mathrm{S} \\
\mathcal{P} &= \{S \to A\,B, \\
& \quad A \to a, \\
& \quad B \to b\,C, \\
& \quad C \to c\,D, \\
& \quad D \to d\}
\end{aligned}
$$

STACK	BUFFER	ACTION
	abcd	SHIFT
a	bcd	REDUCE
A	bcd	SHIFT
Ab	cd	SHIFT
Abc	d	SHIFT
Abcd		REDUCE

Shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$
\begin{aligned}
\Sigma &= \{a, b, c\} \\
\mathcal{N} &= \{S, A, B, C, D\} \\
s &= \mathrm{S} \\
\mathcal{P} &= \{S \rightarrow A\ B, \\
&\quad A \rightarrow a, \\
&\quad B \rightarrow b\ C, \\
&\quad C \rightarrow c\ D, \\
&\quad D \rightarrow d\}
\end{aligned}
$$

STACK	BUFFER	ACTION
	abcd	SHIFT
a	bcd	REDUCE
A	bcd	SHIFT
Ab	cd	SHIFT
Abc	d	SHIFT
Abcd		REDUCE
AbcD		

Shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$
\begin{aligned}
\Sigma &= \{a, b, c\} \\
\mathcal{N} &= \{S, A, B, C, D\} \\
s &= \mathrm{S} \\
\mathcal{P} &= \{S \rightarrow A\ B, \\
&\quad A \rightarrow a, \\
&\quad B \rightarrow b\ C, \\
&\quad C \rightarrow c\ D, \\
&\quad D \rightarrow d\}
\end{aligned}
$$

STACK	BUFFER	ACTION
	abcd	SHIFT
a	bcd	REDUCE
A	bcd	SHIFT
Ab	cd	SHIFT
Abc	d	SHIFT
Abcd		REDUCE
AbcD		REDUCE

Shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$
\begin{aligned}
\Sigma &= \{a, b, c\} \\
\mathcal{N} &= \{S, A, B, C, D\} \\
s &= \mathrm{S} \\
\mathcal{P} &= \{S \rightarrow A\,B, \\
&\quad A \rightarrow a, \\
&\quad B \rightarrow b\,C, \\
&\quad C \rightarrow c\,D, \\
&\quad D \rightarrow d\}
\end{aligned}
$$

STACK	BUFFER	ACTION
	abcd	SHIFT
a	bcd	REDUCE
A	bcd	SHIFT
Ab	cd	SHIFT
Abc	d	SHIFT
Abcd		REDUCE
AbcD		REDUCE
AbC		

Shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$\begin{aligned}
\Sigma &= \{a, b, c\} \\
\mathcal{N} &= \{S, A, B, C, D\} \\
s &= \mathsf{S} \\
\mathcal{P} &= \{S \rightarrow A\ B, \\
&\quad A \rightarrow a, \\
&\quad B \rightarrow b\ C, \\
&\quad C \rightarrow c\ D, \\
&\quad D \rightarrow d\}
\end{aligned}$$

STACK	BUFFER	ACTION
	abcd	SHIFT
a	bcd	REDUCE
A	bcd	SHIFT
Ab	cd	SHIFT
Abc	d	SHIFT
Abcd		REDUCE
AbcD		REDUCE
AbC		REDUCE

Shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$\begin{aligned}
\Sigma &= \{a, b, c\} \\
\mathcal{N} &= \{S, A, B, C, D\} \\
s &= \mathrm{S} \\
\mathcal{P} &= \{S \rightarrow A\,B, \\
&\quad A \rightarrow a, \\
&\quad B \rightarrow b\,C, \\
&\quad C \rightarrow c\,D, \\
&\quad D \rightarrow d\}
\end{aligned}$$

STACK	BUFFER	ACTION
	abcd	SHIFT
a	bcd	REDUCE
A	bcd	SHIFT
Ab	cd	SHIFT
Abc	d	SHIFT
Abcd		REDUCE
AbcD		REDUCE
AbC		REDUCE
AB		

Shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$
\begin{aligned}
\Sigma &= \{a, b, c\} \\
\mathcal{N} &= \{S, A, B, C, D\} \\
s &= \mathrm{S} \\
\mathcal{P} &= \{S \rightarrow A\, B, \\
& \quad A \rightarrow a, \\
& \quad B \rightarrow b\, C, \\
& \quad C \rightarrow c\, D, \\
& \quad D \rightarrow d\}
\end{aligned}
$$

STACK	BUFFER	ACTION
	abcd	SHIFT
a	bcd	REDUCE
A	bcd	SHIFT
Ab	cd	SHIFT
Abc	d	SHIFT
Abcd		REDUCE
AbcD		REDUCE
AbC		REDUCE
AB		REDUCE

Shift-reduce parsing using a **deterministic** CFG

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

$$
\begin{aligned}
\Sigma &= \{a, b, c\} \\
\mathcal{N} &= \{S, A, B, C, D\} \\
s &= \mathrm{S} \\
\mathcal{P} &= \{S \rightarrow A\ B, \\
& \quad A \rightarrow a, \\
& \quad B \rightarrow b\ C, \\
& \quad C \rightarrow c\ D, \\
& \quad D \rightarrow d\}
\end{aligned}
$$

STACK	BUFFER	ACTION
	abcd	SHIFT
a	bcd	REDUCE
A	bcd	SHIFT
Ab	cd	SHIFT
Abc	d	SHIFT
Abcd		REDUCE
AbcD		REDUCE
AbC		REDUCE
AB		REDUCE
S		

Dependency parsers use a **modified** shift-reduce parser

- A common method for dependency parsing of natural language involves a modification of the LR shift-reduce parser
- The **shift** operator continues to move items of the input string from the buffer to the stack
- The **reduce** operator is replaced with the operations **left-arc** and **right-arc** which *reduce* the top two stack symbols leaving the *head* on the stack

Consider $\mathcal{L}(G_{dep}) \subseteq \Sigma^*$, during parsing the stack may hold γab where $\gamma \in \Sigma*$ and $a, b \in \Sigma$, and b is at the top of the stack:

- LEFT-ARC reduces the stack to γb and records use of rule $b \rightarrow a$
- RIGHT-ARC reduces the stack to γa and records the use of rule $a \rightarrow b$

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$
\begin{aligned}
\Sigma &= \{a \ldots z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \mathrm{s} \\
\mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
&\quad (d \rightarrow e, \mathcal{R}) \\
&\quad (e \rightarrow f, \mathcal{L})\}
\end{aligned}
$$

b a c d f e

STACK	BUFFER	ACTION	RECORD
	bacdfe		

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$\begin{aligned}
\Sigma &= \{a...z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \text{s} \\
\mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
&\quad (d \rightarrow e, \mathcal{R}) \\
&\quad (e \rightarrow f, \mathcal{L})\}
\end{aligned}$$

STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	

b a c d f e

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$\begin{aligned}
\Sigma &= \{a...z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \text{s} \\
\mathcal{P} &= \{(a \to b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
&\quad (d \to e, \mathcal{R}) \\
&\quad (e \to f, \mathcal{L})\}
\end{aligned}$$

STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe		

b a c d f e

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$
\begin{aligned}
\Sigma &= \{a...z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \mathrm{s} \\
\mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
& \quad (d \rightarrow e, \mathcal{R}) \\
& \quad (e \rightarrow f, \mathcal{L})\}
\end{aligned}
$$

b a c d f e

STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$
\begin{aligned}
\Sigma &= \{a...z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \mathrm{s} \\
\mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
&\quad (d \rightarrow e, \mathcal{R}) \\
&\quad (e \rightarrow f, \mathcal{L})\}
\end{aligned}
$$

b a c d f e

STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe		

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$\begin{aligned}
\Sigma &= \{a...z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \mathrm{s} \\
\mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
&\quad\ (d \rightarrow e, \mathcal{R}) \\
&\quad\ (e \rightarrow f, \mathcal{L})\}
\end{aligned}$$

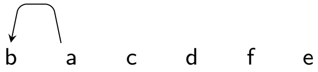
STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \rightarrow b$

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$
\begin{aligned}
\Sigma &= \{a ... z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \text{s} \\
\mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
&\quad (d \rightarrow e, \mathcal{R}) \\
&\quad (e \rightarrow f, \mathcal{L})\}
\end{aligned}
$$

STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \rightarrow b$
a	cdfe		

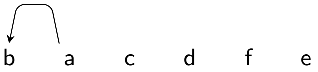


Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$
\begin{aligned}
\Sigma &= \{a...z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \mathrm{s} \\
\mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
& \quad (d \rightarrow e, \mathcal{R}) \\
& \quad (e \rightarrow f, \mathcal{L})\}
\end{aligned}
$$

STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \rightarrow b$
a	cdfe	SHIFT	

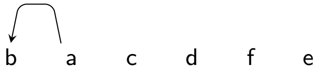


Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$
\begin{aligned}
\Sigma &= \{a...z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \text{s} \\
\mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
&\quad\; (d \rightarrow e, \mathcal{R}) \\
&\quad\; (e \rightarrow f, \mathcal{L})\}
\end{aligned}
$$

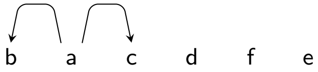
STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \rightarrow b$
a	cdfe	SHIFT	
ac	dfe		



Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$
\begin{array}{lcl}
\Sigma & = & \{a...z\} \\
D & = & \{\mathcal{L}, \mathcal{R}\} \\
s & = & \mathrm{s} \\
\mathcal{P} & = & \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
 & & (d \rightarrow e, \mathcal{R}) \\
 & & (e \rightarrow f, \mathcal{L})\}
\end{array}
$$



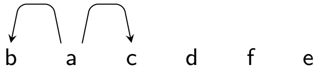
STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \rightarrow b$
a	cdfe	SHIFT	
ac	dfe	RIGHT-ARC	$a \rightarrow c$

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$\begin{aligned}
\Sigma &= \{a \ldots z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \text{s} \\
\mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
&\quad (d \rightarrow e, \mathcal{R}) \\
&\quad (e \rightarrow f, \mathcal{L})\}
\end{aligned}$$

STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \rightarrow b$
a	cdfe	SHIFT	
ac	dfe	RIGHT-ARC	$a \rightarrow c$
a	dfe		

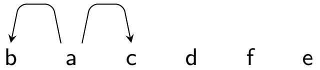


Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$\begin{aligned}
\Sigma &= \{a...z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \text{s} \\
\mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
&\quad\ (d \rightarrow e, \mathcal{R}) \\
&\quad\ (e \rightarrow f, \mathcal{L})\}
\end{aligned}$$

STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \rightarrow b$
a	cdfe	SHIFT	
ac	dfe	RIGHT-ARC	$a \rightarrow c$
a	dfe	SHIFT	

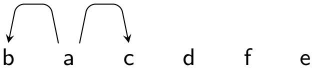


Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$
\begin{aligned}
\Sigma &= \{a...z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \mathrm{s} \\
\mathcal{P} &= \{(a \to b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
& \quad (d \to e, \mathcal{R}) \\
& \quad (e \to f, \mathcal{L})\}
\end{aligned}
$$

STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \to b$
a	cdfe	SHIFT	
ac	dfe	RIGHT-ARC	$a \to c$
a	dfe	SHIFT	
ad	fe		



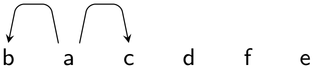
Note that, for a deterministic parse here, a lookahead is needed

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$\begin{aligned}
\Sigma &= \{a...z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \text{s} \\
\mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
&\quad (d \rightarrow e, \mathcal{R}) \\
&\quad (e \rightarrow f, \mathcal{L})\}
\end{aligned}$$

STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \rightarrow b$
a	cdfe	SHIFT	
ac	dfe	RIGHT-ARC	$a \rightarrow c$
a	dfe	SHIFT	
ad	fe	SHIFT	



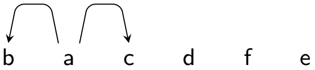
Note that, for a deterministic parse here, a lookahead is needed

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$
\begin{aligned}
\Sigma &= \{a...z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \mathrm{s} \\
\mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
& \quad (d \rightarrow e, \mathcal{R}) \\
& \quad (e \rightarrow f, \mathcal{L})\}
\end{aligned}
$$

STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \rightarrow b$
a	cdfe	SHIFT	
ac	dfe	RIGHT-ARC	$a \rightarrow c$
a	dfe	SHIFT	
ad	fe	SHIFT	
adf	e		

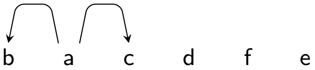


Note that, for a deterministic parse here, a lookahead is needed

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$\begin{aligned} \Sigma &= \{a...z\} \\ D &= \{\mathcal{L}, \mathcal{R}\} \\ s &= \text{s} \\ \mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\ & \quad (d \rightarrow e, \mathcal{R}) \\ & \quad (e \rightarrow f, \mathcal{L})\} \end{aligned}$$



STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \rightarrow b$
a	cdfe	SHIFT	
ac	dfe	RIGHT-ARC	$a \rightarrow c$
a	dfe	SHIFT	
ad	fe	SHIFT	
adf	e	SHIFT	

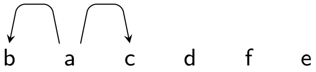
Note that, for a deterministic parse here, a lookahead is needed

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$
\begin{aligned}
\Sigma &= \{a...z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \text{s} \\
\mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
& \quad (d \rightarrow e, \mathcal{R}) \\
& \quad (e \rightarrow f, \mathcal{L})\}
\end{aligned}
$$

STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \rightarrow b$
a	cdfe	SHIFT	
ac	dfe	RIGHT-ARC	$a \rightarrow c$
a	dfe	SHIFT	
ad	fe	SHIFT	
adf	e	SHIFT	
adfe			

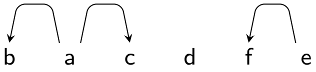


Note that, for a deterministic parse here, a lookahead is needed

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$
\begin{array}{lcl}
\Sigma & = & \{a...z\} \\
D & = & \{\mathcal{L}, \mathcal{R}\} \\
s & = & \mathrm{s} \\
\mathcal{P} & = & \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
 & & (d \rightarrow e, \mathcal{R}) \\
 & & (e \rightarrow f, \mathcal{L})\}
\end{array}
$$



STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \rightarrow b$
a	cdfe	SHIFT	
ac	dfe	RIGHT-ARC	$a \rightarrow c$
a	dfe	SHIFT	
ad	fe	SHIFT	
adf	e	SHIFT	
adfe		LEFT-ARC	$e \rightarrow f$

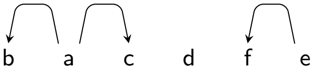
Note that, for a deterministic parse here, a lookahead is needed

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$
\begin{aligned}
\Sigma &= \{a...z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \mathrm{s} \\
\mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
&\quad (d \rightarrow e, \mathcal{R}) \\
&\quad (e \rightarrow f, \mathcal{L})\}
\end{aligned}
$$

STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \rightarrow b$
a	cdfe	SHIFT	
ac	dfe	RIGHT-ARC	$a \rightarrow c$
a	dfe	SHIFT	
ad	fe	SHIFT	
adf	e	SHIFT	
adfe		LEFT-ARC	$e \rightarrow f$
ade			

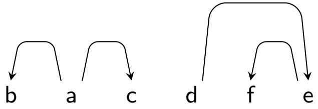


Note that, for a deterministic parse here, a lookahead is needed

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$
\begin{aligned}
\Sigma &= \{a...z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \text{s} \\
\mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
&\quad (d \rightarrow e, \mathcal{R}) \\
&\quad (e \rightarrow f, \mathcal{L})\}
\end{aligned}
$$



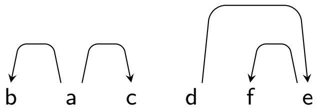
STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \rightarrow b$
a	cdfe	SHIFT	
ac	dfe	RIGHT-ARC	$a \rightarrow c$
a	dfe	SHIFT	
ad	fe	SHIFT	
adf	e	SHIFT	
adfe		LEFT-ARC	$e \rightarrow f$
ade		RIGHT-ARC	$d \rightarrow e$

Note that, for a deterministic parse here, a lookahead is needed

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$
\begin{aligned}
\Sigma &= \{a...z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \mathrm{s} \\
\mathcal{P} &= \{(a \to b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
& \quad (d \to e, \mathcal{R}) \\
& \quad (e \to f, \mathcal{L})\}
\end{aligned}
$$



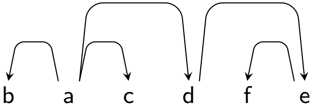
STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \to b$
a	cdfe	SHIFT	
ac	dfe	RIGHT-ARC	$a \to c$
a	dfe	SHIFT	
ad	fe	SHIFT	
adf	e	SHIFT	
adfe		LEFT-ARC	$e \to f$
ade		RIGHT-ARC	$d \to e$
ad			

Note that, for a deterministic parse here, a lookahead is needed

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$\begin{aligned}
\Sigma &= \{a...z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \text{s} \\
\mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
& \quad (d \rightarrow e, \mathcal{R}) \\
& \quad (e \rightarrow f, \mathcal{L})\}
\end{aligned}$$



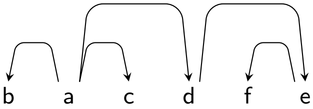
STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \rightarrow b$
a	cdfe	SHIFT	
ac	dfe	RIGHT-ARC	$a \rightarrow c$
a	dfe	SHIFT	
ad	fe	SHIFT	
adf	e	SHIFT	
adfe		LEFT-ARC	$e \rightarrow f$
ade		RIGHT-ARC	$d \rightarrow e$
ad		RIGHT-ARC	$a \rightarrow d$

Note that, for a deterministic parse here, a lookahead is needed

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$
\begin{aligned}
\Sigma &= \{a \ldots z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \text{s} \\
\mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
&\quad (d \rightarrow e, \mathcal{R}) \\
&\quad (e \rightarrow f, \mathcal{L})\}
\end{aligned}
$$



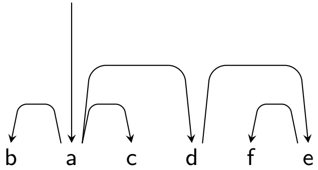
STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \rightarrow b$
a	cdfe	SHIFT	
ac	dfe	RIGHT-ARC	$a \rightarrow c$
a	dfe	SHIFT	
ad	fe	SHIFT	
adf	e	SHIFT	
adfe		LEFT-ARC	$e \rightarrow f$
ade		RIGHT-ARC	$d \rightarrow e$
ad		RIGHT-ARC	$a \rightarrow d$
a			

Note that, for a deterministic parse here, a lookahead is needed

Dependency parsers use a **modified** shift-reduce parser

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, \mathcal{D}, s, \perp, \mathcal{P})$

$$
\begin{aligned}
\Sigma &= \{a...z\} \\
D &= \{\mathcal{L}, \mathcal{R}\} \\
s &= \text{s} \\
\mathcal{P} &= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \\
&\quad (d \rightarrow e, \mathcal{R}) \\
&\quad (e \rightarrow f, \mathcal{L})\}
\end{aligned}
$$



STACK	BUFFER	ACTION	RECORD
	bacdfe	SHIFT	
b	acdfe	SHIFT	
ba	cdfe	LEFT-ARC	$a \rightarrow b$
a	cdfe	SHIFT	
ac	dfe	RIGHT-ARC	$a \rightarrow c$
a	dfe	SHIFT	
ad	fe	SHIFT	
adf	e	SHIFT	
adfe		LEFT-ARC	$e \rightarrow f$
ade		RIGHT-ARC	$d \rightarrow e$
ad		RIGHT-ARC	$a \rightarrow d$
a		TERMINATE	$root \rightarrow a$

Note that, for a deterministic parse here, a lookahead is needed

Data driven dependency parsing is **grammarless**

- For natural language there would be considerable effort in manually defining $\mathcal{P}$—this would involve determining the dependencies between all possible words in the language (although note that e.g. RASP uses a similar transition-based approach with a manually defined PSG)
- Creating a deterministic grammar would be impossible (natural language is inherently ambiguous).
- Natural language dependency parsing can achieved deterministically by **selecting parsing actions** using a machine learning **classifier**.
- The **features** for the classifier include the items in the **configuration** as well as properties of those items (including **word-embeddings** for the items).
- Training is performed on **dependency banks** (that is, sentences that have been manually annotated with their correct dependencies).
- It is said that the parsing is **grammarless**—since no grammar is designed ahead of training.

Data driven dependency parsing is **grammarless**

- For natural language there would be considerable effort in manually defining $\mathcal{P}$—this would involve determining the dependencies between all possible words in the language (although note that e.g. RASP uses a similar transition-based approach with a manually defined PSG)
- Creating a deterministic grammar would be impossible (natural language is inherently ambiguous).
- Natural language dependency parsing can achieved deterministically by **selecting parsing actions** using a machine learning **classifier**.
- The **features** for the classifier include the items in the **configuration** as well as properties of those items (including **word-embeddings** for the items).
- Training is performed on **dependency banks** (that is, sentences that have been manually annotated with their correct dependencies).
- It is said that the parsing is **grammarless**—since no grammar is designed ahead of training.

Data driven dependency parsing is **grammarless**

- For natural language there would be considerable effort in manually defining $\mathcal{P}$—this would involve determining the dependencies between all possible words in the language (although note that e.g. RASP uses a similar transition-based approach with a manually defined PSG)
- Creating a deterministic grammar would be impossible (natural language is inherently ambiguous).
- Natural language dependency parsing can achieved deterministically by **selecting parsing actions** using a machine learning **classifier**.
- The **features** for the classifier include the items in the **configuration** as well as properties of those items (including **word-embeddings** for the items).
- Training is performed on **dependency banks** (that is, sentences that have been manually annotated with their correct dependencies).
- It is said that the parsing is **grammarless**—since no grammar is designed ahead of training.

Data driven dependency parsing is **grammarless**

- For natural language there would be considerable effort in manually defining $\mathcal{P}$—this would involve determining the dependencies between all possible words in the language (although note that e.g. RASP uses a similar transition-based approach with a manually defined PSG)
- Creating a deterministic grammar would be impossible (natural language is inherently ambiguous).
- Natural language dependency parsing can achieved deterministically by **selecting parsing actions** using a machine learning **classifier**.
- The **features** for the classifier include the items in the **configuration** as well as properties of those items (including **word-embeddings** for the items).
- Training is performed on **dependency banks** (that is, sentences that have been manually annotated with their correct dependencies).
- It is said that the parsing is **grammarless**—since no grammar is designed ahead of training.

Data driven dependency parsing is **grammarless**

- For natural language there would be considerable effort in manually defining $\mathcal{P}$—this would involve determining the dependencies between all possible words in the language (although note that e.g. RASP uses a similar transition-based approach with a manually defined PSG)
- Creating a deterministic grammar would be impossible (natural language is inherently ambiguous).
- Natural language dependency parsing can achieved deterministically by **selecting parsing actions** using a machine learning **classifier**.
- The **features** for the classifier include the items in the **configuration** as well as properties of those items (including **word-embeddings** for the items).
- Training is performed on **dependency banks** (that is, sentences that have been manually annotated with their correct dependencies).
- It is said that the parsing is **grammarless**—since no grammar is designed ahead of training.

We can use feature templates to analyse a configuration

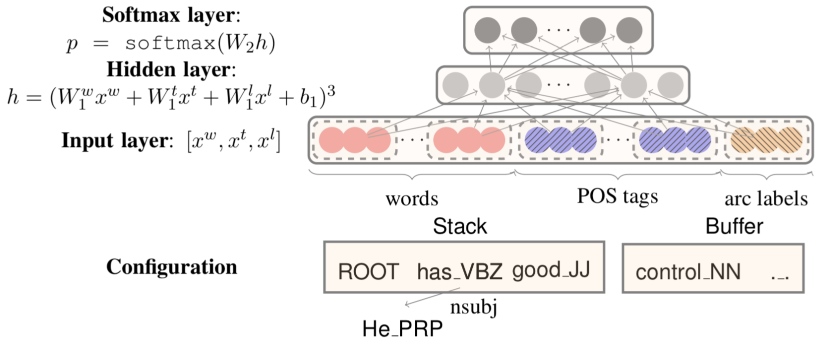
Single-word features (9)
$s_1.w$; $s_1.t$; $s_1.wt$; $s_2.w$; $s_2.t$; $s_2.wt$; $b_1.w$; $b_1.t$; $b_1.wt$
Word-pair features (8)
$s_1.wt \circ s_2.wt$; $s_1.wt \circ s_2.w$; $s_1.wts_2.t$; $s_1.w \circ s_2.wt$; $s_1.t \circ s_2.wt$; $s_1.w \circ s_2.w$ $s_1.t \circ s_2.t$; $s_1.t \circ b_1.t$
Three-word feaures (8)
$s_2.t \circ s_1.t \circ b_1.t$; $s_2.t \circ s_1.t \circ lc_1(s_1).t$; $s_2.t \circ s_1.t \circ rc_1(s_1).t$; $s_2.t \circ s_1.t \circ lc_1(s_2).t$; $s_2.t \circ s_1.t \circ rc_1(s_2).t$; $s_2.t \circ s_1.w \circ rc_1(s_2).t$; $s_2.t \circ s_1.w \circ lc_1(s_1).t$; $s_2.t \circ s_1.w \circ b_1.t$

Chen and Manning, ACL, 2014

There are problems with the features

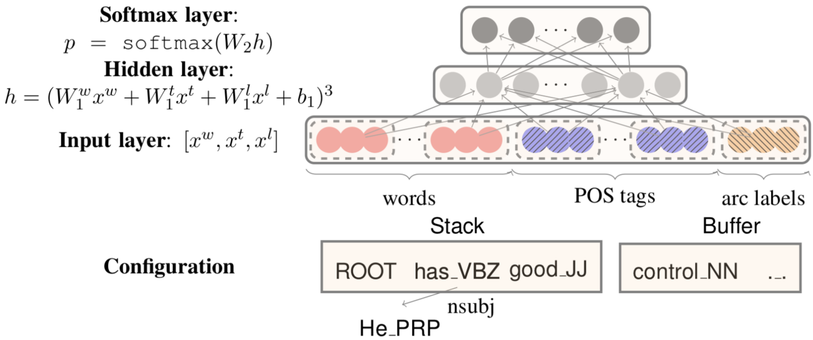
- The features are indispensable but **highly sparse**.
- The feature templates are **incomplete**.
- The features are **expensive to compute**.

Chen and Manning present NN dependency parser in 2014



Consider sets S^w, S^t, S^l:
e.g. $S^t = \{lc_1(s_2).t, s_2.t, rc_1(s_2).t, s_1.t\} \rightarrow PRP, VBZ, NULL, JJ$
and w^t is the concatenation of the tag embeddings.

Chen and Manning present NN dependency parser in 2014



Consider sets S^w, S^t, S^l:
e.g. $S^t = \{lc_1(s_2).t, s_2.t, rc_1(s_2).t, s_1.t\} \rightarrow PRP, VBZ, NULL, JJ$
and w^t is the concatenation of the tag embeddings.

Chen and Manning use a rich feature sets

- S^w contains 18 feature templates:
  1. The top 3 words on the stack and buffer

     $s_1, s_2, s_3, b_1, b_2, b_3$
  2. The first and second leftmost and rightmost children of the top two words on the stack

     $lc_1(s_i), rc_1(s_i), lc_2(s_i), rc_2(s_i)$ where $i = 1, 2$
  3. The leftmost of leftmost and rightmost of rightmost children of the top two words on the stack.

     $lc_1(lc_1(s_i)), rc_1(rc_1(s_i))$ where $i = 1, 2$

- S^T is the corresponding tags
- S^l the corresponding arc labels (excludes category 1 above)

Chen and Manning use a rich feature sets

- S^w contains 18 feature templates:
  1. The top 3 words on the stack and buffer

     $s_1, s_2, s_3, b_1, b_2, b_3$
  2. The first and second leftmost and rightmost children of the top two words on the stack

     $lc_1(s_i), rc_1(s_i), lc_2(s_i), rc_2(s_i)$ where $i = 1, 2$
  3. The leftmost of leftmost and rightmost of rightmost children of the top two words on the stack.

     $lc_1(lc_1(s_i)), rc_1(rc_1(s_i))$ where $i = 1, 2$

- S^T is the corresponding tags
- S^l the corresponding arc labels (excludes category 1 above)

Details of the original Chen and Manning

- Generate training examples (pairing real configurations with their gold parsing actions) from dependancy bank using a shortest stack oracle
- Training objective is to minimize cross-entropy loss
- Back-propagation to the embeddings during training

- During parsing they use greedy decoding.

- Further improvements in recent years.