Formal Models of Language

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge

Recap:

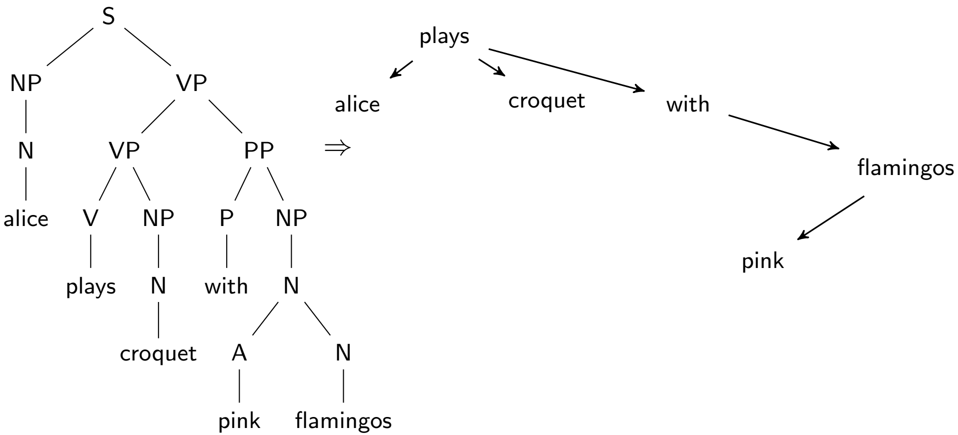
- We said LR shift-reduce parser wasn't a good fit for natural language because it proceeds deterministically and natural language is too ambiguous.
- We used the Earley parser to explore the whole tree-space, recording partial derivations in a chart.

However,

- We can use a modified version of the shift-reduce parser in order to parse natural language.
- First we're going to learn about **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** derives **dependency trees**

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

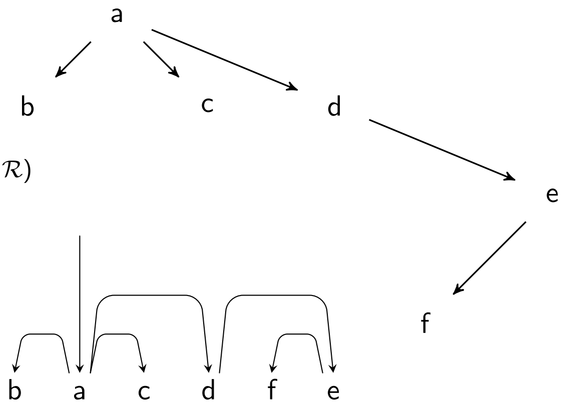
- Σ is the finite set of alphabet symbols
- $\mathcal{D}$ is the set of symbols to indicate whether the dependent symbol (the one on the RHS of the rule) will be located on the left or right of the current item within the string $\mathcal{D} = \{\mathcal{L}, \mathcal{R}\}$
- s is the root symbol for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

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:

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



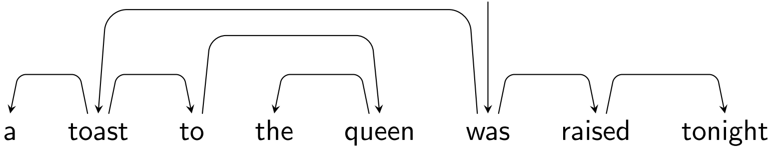
The same rules would have been used to generate the string *badfec*.

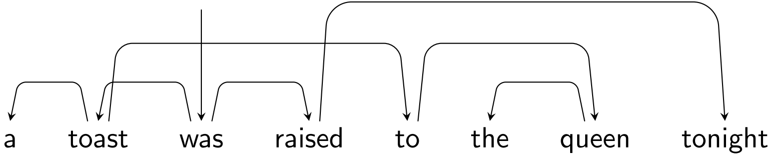
Useful when there is flexibility in the symbol order of grammatical strings.

Valid trees may be **projective** or **non-projective**

Valid derivation is one that is **rooted** in s and is **weakly connected**.

- Derivation trees may be **projective** or **non-projective**.
- Non-projective trees can be needed for long distance dependencies.





- The difference has implications for parsing complexity.

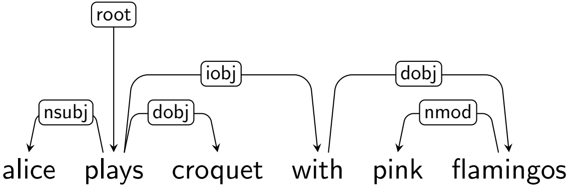
Labels can be added to the dependency edges

A label can be added to each generated dependency:

$$\mathcal{P} = \{(\alpha \to \beta \ : r,\ d) \mid \alpha \in (\Sigma \ \cup \ s),\ \beta \in (\Sigma \cup \perp),\ d \in \mathcal{D},\ r \in \mathcal{B}\}$$

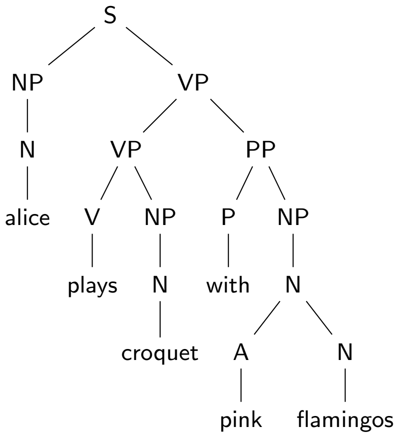
where $\mathcal{B}$ is the set of dependency labels.

When used for natural language parsing, dependency grammars will often label each dependency with the *grammatical function* (or the **grammatical relation**) between the words.

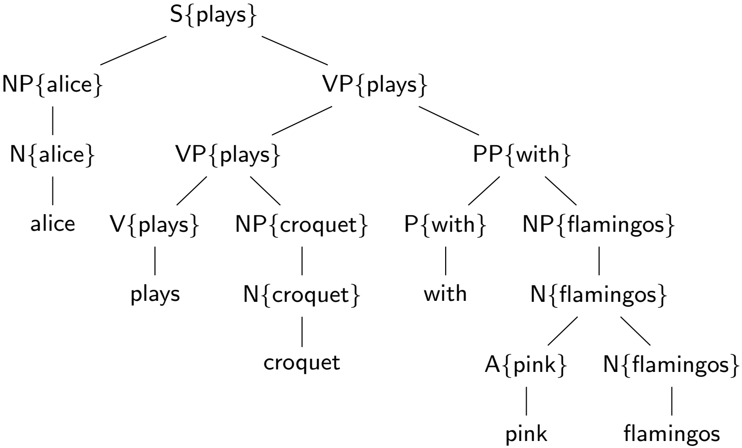


Dependency grammars can be **weakly equivalent** to CFGs

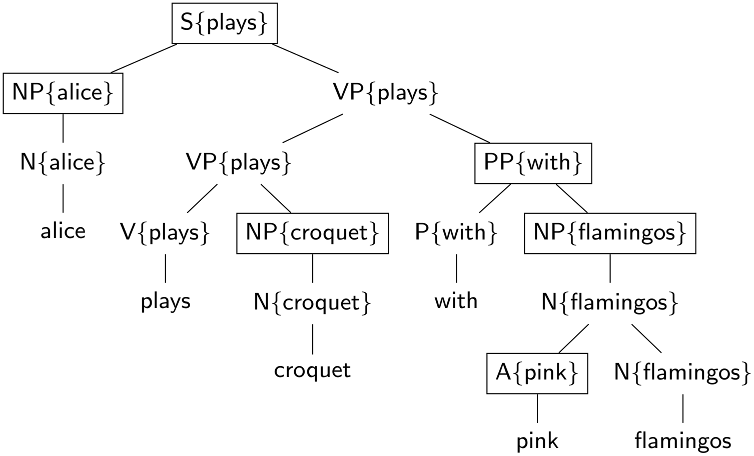
Projective dependency grammars can be shown to be **weakly equivalent** to context-free grammars.



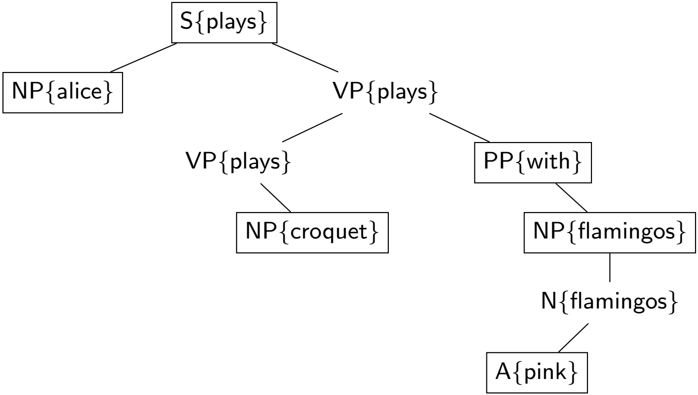
Dependency grammars can be **weakly equivalent** to CFGs



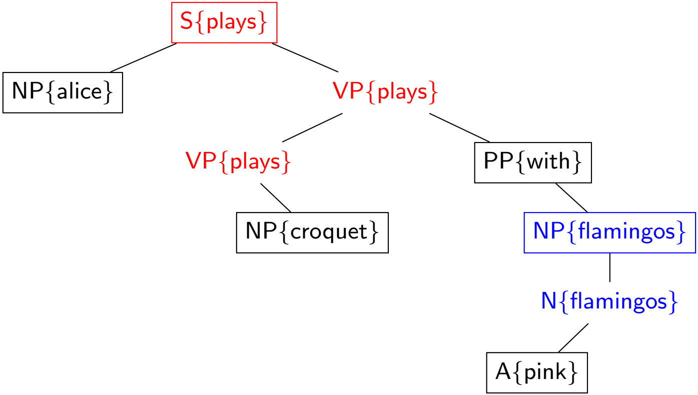
Dependency grammars can be **weakly equivalent** to CFGs



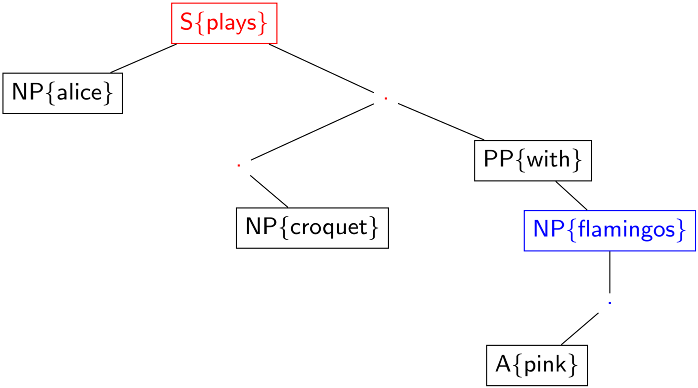
Dependency grammars can be **weakly equivalent** to CFGs



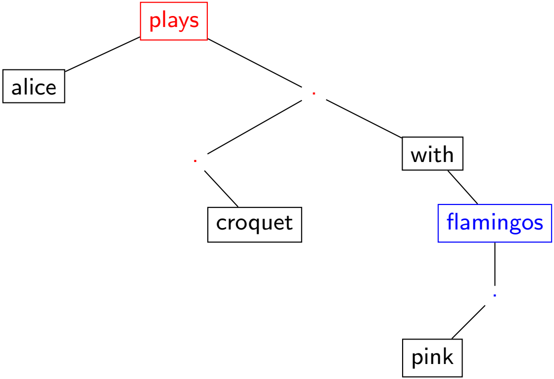
Dependency grammars can be **weakly equivalent** to CFGs



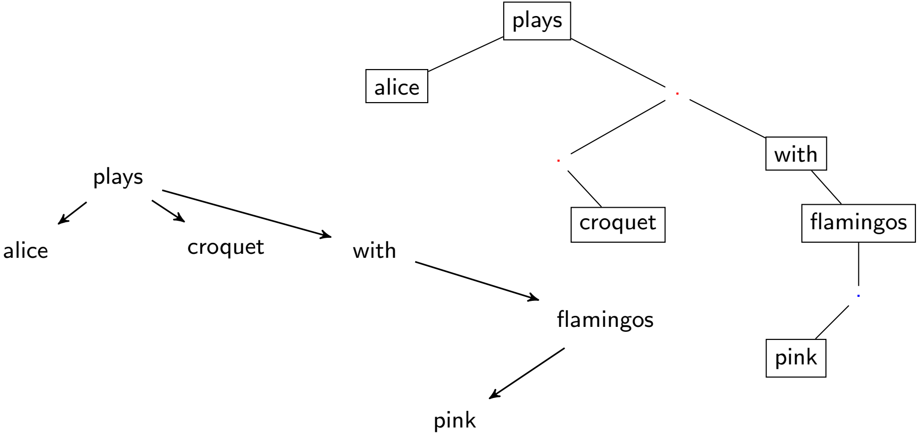
Dependency grammars can be **weakly equivalent** to CFGs



Dependency grammars can be **weakly equivalent** to CFGs



Dependency grammars can be **weakly equivalent** to CFGs



Projective dependency grammars can be shown to be **weakly equivalent** to context-free grammars.

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

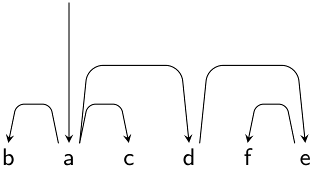
Consider $\mathcal{L}(G_{dep}) \subseteq \Sigma^*$, during parsing the stack may hold γab where $\gamma \in \Sigma*$ and $a, b \in \Sigma$, and b is at the top of the stack:

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

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



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.
- Creating a deterministic grammar would be impossible (natural language is inherently ambiguous).
- Natural language dependency parsing can be achieved deterministically by **selecting parsing actions** using a machine learning **classifier**.
- The **features** for the classifier include the items on the stack and in the buffer 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 a **beam** search to record the parse forest

- The classifier can return a **probability** of an action.
- To avoid the problem of early incorrect resolution of an ambiguous parse, multiple competing parses can be recorded and a **beam search** used to keep track of the best alternative parses.
- Google's *Parsey McParseface* is an English language dependency parser that uses word-embeddings as features and a neural network to score parse actions. A beam search is used to compare competing parses.

Dependency parsers can be useful for parsing **speech**

The most obvious difference between spoken and written language is the mode of transmission:

- **Prosody** refers to the patterns of stress and intonation in a language.
- **Stress** refers to the relative emphasis or prominence given to a certain part of a word (e.g. CON-tent (the stuff included in something) vs. con-TENT (happy))
- **Intonation** refers to the way speakers' pitch rises and falls in line with words and phrases, to signal a question, for example.
- Co-speech gestures involve parts of the body which move in coordination with what a speaker is saying, to emphasise, disambiguate or otherwise.

We can use some of these extra features to help the parse-action-classifier when parsing spoken language.

Prosody has been used to resolve parsing ambiguity

- Briscoe suggested using a shift-reduce parser that favours shift over reduce wherever both are possible.
- In the absence of extra-linguistic information the parser delays resolution of the grammatical dependency.
- Extra features enable an override of the shift preference at the point where the ambiguity arises, including:
  - prosodic information (intonational phrase boundary)

  The model accounts for frequencies of certain syntactic constructions as attested in corpora.

Spoken language **lacks** string delimitation

- A fundamental issue that affects syntactic parsing of spoken language is the **lack of the sentence unit** (i.e string delimitation)—indicated in writing by a full-stop and capital letter.
- **Speech units** may be identified by pauses, intonation (e.g. rising for a question, falling for a full-stop), change of speaker.
- Speech units are not much like written sentences due to **speaker overlap**, **co-constructions**, **ellipsis**, **hesitation**, **repetitions** and **false starts**.
- Speech units often contain words and grammatical constructions that would not appear in the written form of the language.

Spoken language **lacks** string delimitation

Excerpt from the Spoken section of the British National Corpus

set your sights realistically haven't you and there's a lot of people unemployed and what are you going to do when you eventually leave college if you get there you're not gonna step straight into television mm right then let's see now what we're doing where's that recipe book for that chocolate and banana cake chocolate and banana cake which book was it oh right oh some of these chocolate cakes are absolutely mm mm mm right what's the topping what's that icing sugar cocoa powder and vanilla essence oh luckily I've got all those I think yes

Spoken language **lacks** string delimitation

Excerpt from the Spoken section of the British National Corpus

Set your sights realistically haven't you? And there's a lot of people unemployed. And what are you going to do when you eventually leave college? If you get there. You're not gonna step straight into television. Mm right then, let's see now what we're doing... Where's that recipe book for that chocolate and banana cake? Chocolate and banana cake which book was it? Oh right. Oh, some of these chocolate cakes are absolutely mm mm mm. Right, what's the topping? what's that? Icing sugar, cocoa powder and vanilla essence. Oh luckily I've got all those I think, yes!

Dependency parsers can be useful for parsing **speech**

- Spoken language can look noisy and somewhat *grammarless* but the disfluencies are predictable
- Honnibal & Johnson's Redshift parser introduces an **edit** action, to remove disfluent items from spoken language:
    - EDIT: on detection of disfluency, remove connected words and their dependencies.
- Parser uses extra classifier features to detect disfluency.

Example of dependency parser using an **edit** action

STACK	BUFFER	ACTION	RECORD
	his_1 ... $bankrupt_7$	SHIFT	
his_1	$company_2$ ... $bankrupt_7$	SHIFT	
his_1 $company_2$	$went_3$ ... $bankrupt_7$	LEFT-ARC	$company_2 \rightarrow his_1$
$company_2$	$went_3$ ... $bankrupt_7$	SHIFT	
$company_2$ $went_3$	$broke_4$ ... $bankrupt_7$	LEFT-ARC	$\cancel{went_3 \rightarrow company_2}$
$went_3$	$broke_4$ ... $bankrupt_7$	SHIFT	
$went_3$ $broke_4$	$I - mean_5$ ... $bankrupt_7$	RIGHT-ARC	$\cancel{went_3 \rightarrow broke_4}$
$went_3$	$I - mean_5$ ... $bankrupt_7$	SHIFT	
$went_3$ $I - mean_5$	$went_6$ $bankrupt_7$	EDIT	
$company_2$	$went_6$ $bankrupt_7$	SHIFT	
$company_2$ $went_6$	$bankrupt_7$	LEFT-ARC	$went_6 \rightarrow company_2$
$went_6$	$bankrupt_7$	SHIFT	
$went_3$ $bankrupt_7$		RIGHT-ARC	$went_6 \rightarrow bankrupt_7$
$went_3$		TERMINATE	$root \rightarrow went_6$