L95: Natural Language Syntax and Parsing
2) PCFGs and CKY parsing

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge

Reminder: languages can also be defined using **automata**

Recall that a language is regular if it is equal to the set of strings accepted by some deterministic finite-state automaton (DFA).

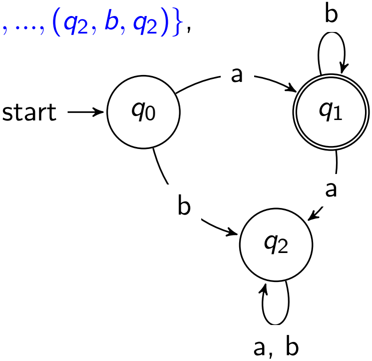
A DFA is defined as $M = (\mathcal{Q}, \Sigma, \Delta, s, \mathcal{F})$ where:

- $\mathcal{Q} = \{q_0, q_1, q_2...\}$ is a finite set of states.
- Σ is the alphabet: a finite set of transition symbols.
- $\Delta \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q}$ is a function $\mathcal{Q} \times \Sigma \rightarrow \mathcal{Q}$ which we write as δ. Given $q \in \mathcal{Q}$ and $i \in \Sigma$ then $\delta(q, i)$ returns a new state $q' \in \mathcal{Q}$
- s is a starting state
- $\mathcal{F}$ is the set of all end states

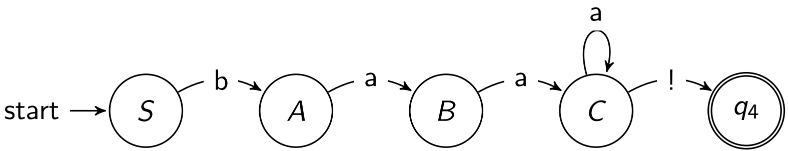
Reminder: **regular languages** are accepted by **DFAs**

For $\mathcal{L}(M) = \{a, ab, abb, ...\}$:

M=($\mathcal{Q} = \{q_0, q_1, q_2\}$,
$\Sigma = \{a, b\}$,
$\Delta = \{(q_0, a, q_1), (q_0, b, q_2), ..., (q_2, b, q_2)\}$,
$s = q_0$,
$\mathcal{F} = \{q_1\}$)



Simple relationship between a DFA and **production rules**



$$Q = \{S, A, B, C, q_4\}$$
$$\Sigma = \{b, a, !\}$$
$$q_0 = S$$
$$F = \{q_4\}$$

$$S \rightarrow bA$$
$$A \rightarrow aB$$
$$B \rightarrow aC$$
$$C \rightarrow aC$$
$$C \rightarrow !$$

Regular grammars generate regular languages

Given a DFA $M = (\mathcal{Q}, \Sigma, \Delta, s, \mathcal{F})$ the language, $\mathcal{L}(M)$, of strings accepted by M can be generated by the regular grammar $G_{reg} = (\mathcal{N}, \Sigma, S, \mathcal{P})$ where:

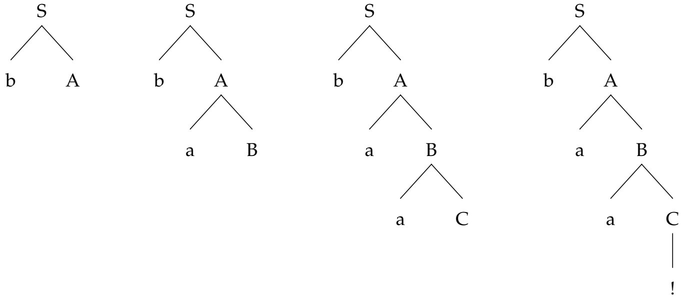
- $\mathcal{N} = \mathcal{Q}$ the non-terminals are the states of M
- $\Sigma = \Sigma$ the terminals are the set of transition symbols of M
- $S = s$ the starting symbol is the starting state of M
- $\mathcal{P} = q_i \rightarrow aq_j$ when $\delta(q_i, a) = q_j \in \Delta$
  or $q_i \rightarrow \epsilon$ when $q \in \mathcal{F}$ (i.e. when q is an end state)

Strings are **derived** from production rules

In order to derive a string from a grammar

- start with the designated starting symbol
- then non-terminal symbols are repeatedly expanded using the rewrite rules until there is nothing further left to expand.

The rewrite rules derive the members of a language from their internal structure (or **phrase structure**)



$S \rightarrow bA$ $A \rightarrow aB$ $B \rightarrow aC$ $C \rightarrow !$

A regular language has a **left-** and **right-linear** grammar

For every regular grammar the rewrite rules of the grammar can all be expressed in the form:

$$X \rightarrow aY$$
$$X \rightarrow a$$

or alternatively, they can all be expressed as:

$$X \rightarrow Ya$$
$$X \rightarrow a$$

The two grammars are **weakly-equivalent** since they generate the same strings.
But not **strongly-equivalent** because they do not generate the same structure to strings

A regular language has a **left-** and **right-linear** grammar

For every regular grammar the rewrite rules of the grammar can all be expressed in the form:

$$X \rightarrow aY$$
$$X \rightarrow a$$

or alternatively, they can all be expressed as:

$$X \rightarrow Ya$$
$$X \rightarrow a$$

The two grammars are **weakly-equivalent** since they generate the same strings.

But not **strongly-equivalent** because they do not generate the same structure to strings

A regular language has a **left-** and **right-linear** grammar

For every regular grammar the rewrite rules of the grammar can all be expressed in the form:

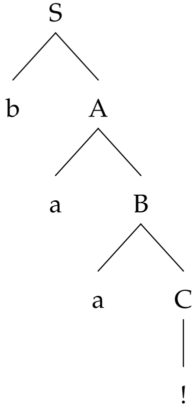
$$X \rightarrow aY$$
$$X \rightarrow a$$

or alternatively, they can all be expressed as:

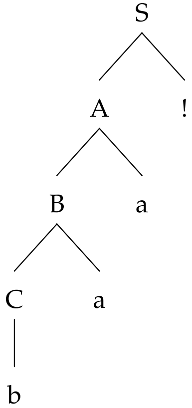
$$X \rightarrow Ya$$
$$X \rightarrow a$$

The two grammars are **weakly-equivalent** since they generate the same strings.
But not **strongly-equivalent** because they do not generate the same structure to strings

A regular language has a **left-** and **right-linear** grammar



$$
\begin{array}{lcl}
S & \rightarrow & bA \\
A & \rightarrow & aB \\
B & \rightarrow & aC \\
C & \rightarrow & aC \\
C & \rightarrow & !
\end{array}
$$



$$
\begin{array}{lcl}
S & \rightarrow & A! \\
A & \rightarrow & Ba \\
B & \rightarrow & Ca \\
C & \rightarrow & Ca \\
C & \rightarrow & b
\end{array}
$$

A regular grammar is a **phrase structure grammar**

A phrase structure grammar over an alphabet Σ is defined by a tuple $G = (\mathcal{N}, \Sigma, S, \mathcal{P})$. The language generated by grammar G is $\mathcal{L}(G)$:

NON-TERMINALS $\mathcal{N}$: Non-terminal symbols (often uppercase letters) may be **rewritten** using the rules of the grammar.

TERMINALS Σ: Terminal symbols (often lowercase letters) are elements of Σ and **cannot be rewritten**. Note $\mathcal{N} \cap \Sigma = \emptyset$.

START SYMBOL S: A **distinguished non-terminal symbol** $S \in \mathcal{N}$. This non-terminal provides the starting point for derivations.

PHRASE STRUCTURE RULES $\mathcal{P}$: Phrase structure rules are pairs of the form (w, v) usually written:
$w \to v$, where $w \in (\Sigma \cup \mathcal{N})^* \mathcal{N} (\Sigma \cup \mathcal{N})^*$ and $v \in (\Sigma \cup \mathcal{N})^*$

Definition of a phrase structure grammar **derivation**

Given $G = (\mathcal{N}, \Sigma, S, \mathcal{P})$ and $w, v \in (\mathcal{N} \cup \Sigma)^*$ a **derivation step** is possible to transform w into v if:

$u_1, u_2 \in (\mathcal{N} \cup \Sigma)^*$ exist such that $w = u_1 \alpha u_2$, and $v = u_1 \beta u_2$ and $\alpha \to \beta \in \mathcal{P}$

This is written $w \underset{G}{\Rightarrow} v$

A string in the language $\mathcal{L}(G)$ is a member of Σ^* that can be derived in a **finite number of derivation steps** from the starting symbol S.

We use $\underset{G^*}{\Longrightarrow}$ to denote the reflexive, transitive closure of derivation steps, consequently $\mathcal{L}(G) = \{w \in \Sigma^* | S \underset{G^*}{\Longrightarrow} w\}$.

PSGs may be grouped by production rule properties

Chomsky suggested that phrase structure grammars may be grouped together by the properties of their production rules.

Name	Form of Rules
regular	$(A \rightarrow Aa$ or $A \rightarrow aA)$ and $A \rightarrow a \mid A \in \mathcal{N}$ and $a \in \Sigma$
context-free	$A \rightarrow \alpha \mid A \in \mathcal{N}$ and $\alpha \in (\mathcal{N} \cup \Sigma)^*$
context-sensitive	$\alpha A \beta \rightarrow \alpha \gamma \beta \mid A \in \mathcal{N}$ and $\alpha, \beta, \gamma \in (\mathcal{N} \cup \Sigma)^*$ and $\gamma \neq \epsilon$
recursively enum	$\alpha \rightarrow \beta \mid \alpha, \beta \in (\mathcal{N} \cup \Sigma)^*$ and $\alpha \neq \epsilon$

A **class** of languages (e.g. the class of regular languages) is all the languages that can be generated by a particular TYPE of grammar.

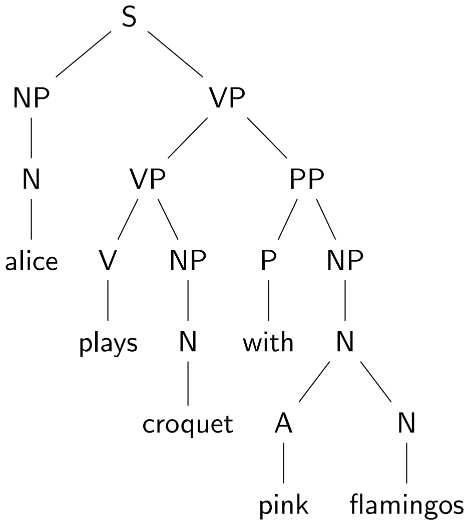
The term **power** is used to describe the **expressivity** of each type of grammar in the hierarchy (measured in terms of the number of subsets of Σ^* that the type can generate)

We can define the **complexity** of language classes

The **complexity** of a language class is defined in terms of the **recognition problem**.

Type	Language Class	Complexity
3	regular	$O(n)$
2	context-free	$O(n^c)$
1	context-sensitive	$O(c^n)$
0	recursively enumerable	*undecidable*

Context-free grammars capture **constituency**



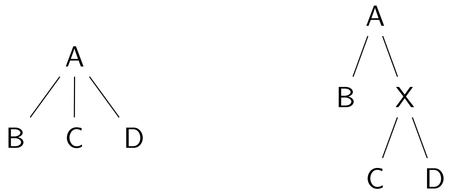
$G = (\mathcal{N}, \Sigma, S, \mathcal{P})$ where
$\mathcal{P} = \{A \rightarrow \alpha \mid$
$A \in \mathcal{N}, \alpha \in (\mathcal{N} \cup \Sigma)^*\}$

CFGs can be written in **Chomsky Normal Form**

Chomsky normal form: every production rule has the form, $A \rightarrow BC$, or, $A \rightarrow a$ where $A, B, C \in \mathcal{N}$, and, $a \in \Sigma$.

Conversion to Chomsky Normal Form

For every CFG there is a weakly equivalent CNF alternative.
$A \rightarrow BCD$ may be rewritten as the two rules, $A \rightarrow BX$, and, $X \rightarrow CD$.



CFGs can be written in **Chomsky Normal Form**

For $A, B, C, D, X, Y \in \mathcal{N}$ and $\gamma, \beta \subseteq \mathcal{N}*$ and $a \in \Sigma$.

Conversion to Chomsky Normal Form

- Keep all existing conforming rules
- replace $A \rightarrow \gamma a \beta$ with $D \rightarrow \gamma A \beta$ and $A \rightarrow a$
- repeatedly replace $A \rightarrow \gamma BC$ with $A \rightarrow \gamma X$ and $X \rightarrow BC$
- if $A \underset{*}{\Rightarrow} B$ is a chain of one or more unit productions, and $B \rightarrow a$ then replace all the unit productions with $A \rightarrow a$ (where a unit production is any rule of the form $X \rightarrow Y$)

CNF is a requirement for the CKY parsing algorithm but it causes some problems:

- Grammar is no longer linguistically intuitive
- Direct correspondence with compositional semantics may be lost

CFGs can be written in **Chomsky Normal Form**

For $A, B, C, D, X, Y \in \mathcal{N}$ and $\gamma, \beta \subseteq \mathcal{N}*$ and $a \in \Sigma$.

Conversion to Chomsky Normal Form

- Keep all existing conforming rules
- replace $A \rightarrow \gamma a \beta$ with $D \rightarrow \gamma A \beta$ and $A \rightarrow a$
- repeatedly replace $A \rightarrow \gamma BC$ with $A \rightarrow \gamma X$ and $X \rightarrow BC$
- if $A \underset{*}{\Rightarrow} B$ is a chain of one or more unit productions, and $B \rightarrow a$ then replace all the unit productions with $A \rightarrow a$ (where a unit production is any rule of the form $X \rightarrow Y$)

CNF is a requirement for the CKY parsing algorithm but it causes some problems:

- Grammar is no longer linguistically intuitive
- Direct correspondence with compositional semantics may be lost

CFGs used to model natural language are **not** deterministic

Deterministic context-free languages:

- are a proper subset of the context-free languages
- can be modelled by an unambiguous grammar
- can be parsed in linear time
- parser can be automatically generated from the grammar

CFGs used to model natural language are **not** deterministic

- Natural languages (with all their inherent ambiguity) are not well suited to algorithms which operate deterministically recognising a single derivation without backtracking
- However, natural language parsing can be achieved deterministically by selecting parsing actions using a machine learning classifier (more on this in later lectures).
- All CFLs (including those exhibiting ambiguity) can be recognised in polynomial time using **dynamic programming algorithms**.

CFGs used to model natural language are **not** deterministic

Deterministic context-free languages:

- are a proper subset of the context-free languages
- can be modelled by an unambiguous grammar
- can be parsed in linear time
- parser can be automatically generated from the grammar

CFGs used to model natural language are **not** deterministic

- Natural languages (with all their inherent ambiguity) are not well suited to algorithms which operate deterministically recognising a single derivation without backtracking
- However, natural language parsing can be achieved deterministically by selecting parsing actions using a machine learning classifier (more on this in later lectures).
- All CFLs (including those exhibiting ambiguity) can be recognised in polynomial time using **dynamic programming algorithms**.

CFGs used to model natural language are **not** deterministic

Deterministic context-free languages:

- are a proper subset of the context-free languages
- can be modelled by an unambiguous grammar
- can be parsed in linear time
- parser can be automatically generated from the grammar

CFGs used to model natural language are **not** deterministic

- Natural languages (with all their inherent ambiguity) are not well suited to algorithms which operate deterministically recognising a single derivation without backtracking
- However, natural language parsing can be achieved deterministically by selecting parsing actions using a machine learning classifier (more on this in later lectures).
- All CFLs (including those exhibiting ambiguity) can be recognised in polynomial time using **dynamic programming algorithms**.

CFGs used to model natural language are **not** deterministic

Deterministic context-free languages:

- are a proper subset of the context-free languages
- can be modelled by an unambiguous grammar
- can be parsed in linear time
- parser can be automatically generated from the grammar

CFGs used to model natural language are **not** deterministic

- Natural languages (with all their inherent ambiguity) are not well suited to algorithms which operate deterministically recognising a single derivation without backtracking
- However, natural language parsing can be achieved deterministically by selecting parsing actions using a machine learning classifier (more on this in later lectures).
- All CFLs (including those exhibiting ambiguity) can be recognised in polynomial time using **dynamic programming algorithms**.

The CKY algorithm **recognises** strings in a CFL

0 they 1 can 2 fish 3

		TO	
	1	2	3
0			
FROM 1			
2			
	they	*can*	*fish*

Toy CNF grammar:

$$\mathcal{N} = \{S, NP, VP, VV, VM\}$$
$$\Sigma = \{can, fish, they\}$$
$$S = S$$
$$\mathcal{P} = \{S \rightarrow NP\ VP$$
$$VP \rightarrow VM\ VV$$
$$VP \rightarrow VV\ NP$$
$$VV \rightarrow can \mid fish$$
$$VM \rightarrow can$$
$$NP \rightarrow they \mid fish\ \}$$

String is in the language when the cell $[0, 3]$ contains S

The CKY algorithm **recognises** strings in a CFL

0 they 1 can 2 fish 3

		TO		
		1	2	3
	0			
FROM	1			
	2			
		they	*can*	*fish*

Toy CNF grammar:

$$\begin{aligned}
\mathcal{N} &= \{S, NP, VP, VV, VM\} \\
\Sigma &= \{can, fish, they\} \\
S &= S \\
\mathcal{P} &= \{S \rightarrow NP\ VP \\
&\quad VP \rightarrow VM\ VV \\
&\quad VP \rightarrow VV\ NP \\
&\quad VV \rightarrow can \mid fish \\
&\quad VM \rightarrow can \\
&\quad NP \rightarrow they \mid fish\ \}
\end{aligned}$$

String is in the language when the cell $[0, 3]$ contains S

The CKY algorithm **recognises** strings in a CFL

0 they 1 can 2 fish 3

		TO 1	2	3
	0	*NP*		
FROM	1			
	2			
		they	*can*	*fish*

Toy CNF grammar:

$$\begin{aligned}
\mathcal{N} &= \{S, NP, VP, VV, VM\} \\
\Sigma &= \{can, fish, they\} \\
S &= S \\
\mathcal{P} &= \{S \rightarrow NP\ VP \\
&\quad VP \rightarrow VM\ VV \\
&\quad VP \rightarrow VV\ NP \\
&\quad VV \rightarrow can \mid fish \\
&\quad VM \rightarrow can \\
&\quad NP \rightarrow they \mid fish\ \}
\end{aligned}$$

String is in the language when the cell $[0, 3]$ contains S

The CKY algorithm **recognises** strings in a CFL

0 they 1 can 2 fish 3

FROM \ TO	1	2	3
0	*NP*		
1		*VV* *VM*	
2			
	they	*can*	*fish*

Toy CNF grammar:

$$
\begin{aligned}
\mathcal{N} &= \{S, NP, VP, VV, VM\} \\
\Sigma &= \{can, fish, they\} \\
S &= S \\
\mathcal{P} &= \{S \rightarrow NP\ VP \\
&\quad VP \rightarrow VM\ VV \\
&\quad VP \rightarrow VV\ NP \\
&\quad VV \rightarrow can \mid fish \\
&\quad VM \rightarrow can \\
&\quad NP \rightarrow they \mid fish\ \}
\end{aligned}
$$

String is in the language when the cell $[0, 3]$ contains S

The CKY algorithm **recognises** strings in a CFL

0 they 1 can 2 fish 3

		TO 1	2	3
	0	*NP*	·	
FROM	1		*VV* *VM*	
	2			
		they	*can*	*fish*

Toy CNF grammar:

$$\begin{aligned}
\mathcal{N} &= \{S, NP, VP, VV, VM\} \\
\Sigma &= \{can, fish, they\} \\
S &= S \\
\mathcal{P} &= \{S \rightarrow NP\ VP \\
&\quad VP \rightarrow VM\ VV \\
&\quad VP \rightarrow VV\ NP \\
&\quad VV \rightarrow can \mid fish \\
&\quad VM \rightarrow can \\
&\quad NP \rightarrow they \mid fish\ \}
\end{aligned}$$

String is in the language when the cell $[0, 3]$ contains S

The CKY algorithm **recognises** strings in a CFL

0 they 1 can 2 fish 3

FROM \ TO	1	2	3
0	*NP*	·	
1		*VV* *VM*	
2			*VV* *NP*
	they	*can*	*fish*

Toy CNF grammar:

$$
\begin{aligned}
\mathcal{N} &= \{S, NP, VP, VV, VM\} \\
\Sigma &= \{can, fish, they\} \\
S &= S \\
\mathcal{P} &= \{S \rightarrow NP\ VP \\
& \quad VP \rightarrow VM\ VV \\
& \quad VP \rightarrow VV\ NP \\
& \quad VV \rightarrow can \mid fish \\
& \quad VM \rightarrow can \\
& \quad NP \rightarrow they \mid fish\ \}
\end{aligned}
$$

String is in the language when the cell $[0, 3]$ contains S

The CKY algorithm **recognises** strings in a CFL

0 they 1 can 2 fish 3

FROM \ TO	1	2	3
0	*NP*	·	
1		*VV* *VM*	*VP*
2			*VV* *NP*
	they	*can*	*fish*

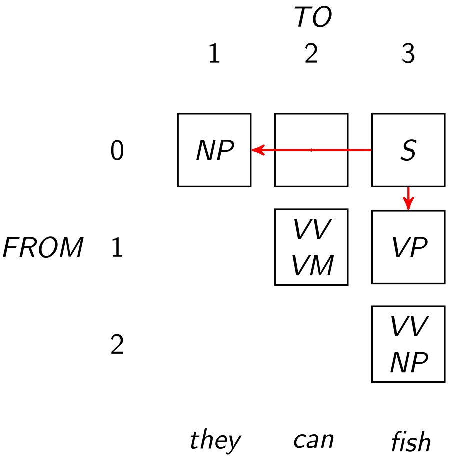
Toy CNF grammar:

$$
\begin{aligned}
\mathcal{N} &= \{S, NP, VP, VV, VM\} \\
\Sigma &= \{can, fish, they\} \\
S &= S \\
\mathcal{P} &= \{S \rightarrow NP\ VP \\
&\quad VP \rightarrow VM\ VV \\
&\quad VP \rightarrow VV\ NP \\
&\quad VV \rightarrow can \mid fish \\
&\quad VM \rightarrow can \\
&\quad NP \rightarrow they \mid fish\ \}
\end{aligned}
$$

String is in the language when the cell $[0,3]$ contains S

The CKY algorithm **recognises** strings in a CFL

0 they 1 can 2 fish 3



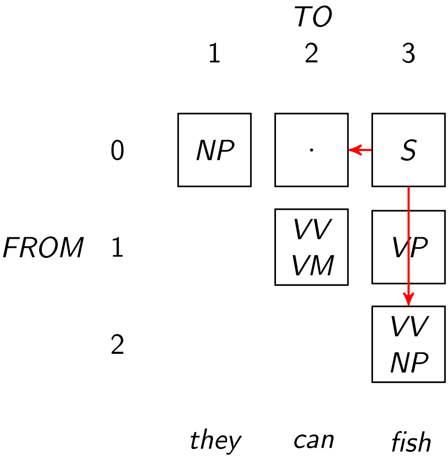
Toy CNF grammar:

$$
\begin{aligned}
\mathcal{N} &= \{S, NP, VP, VV, VM\} \\
\Sigma &= \{can, fish, they\} \\
S &= S \\
\mathcal{P} &= \{S \rightarrow NP\ VP \\
&\quad VP \rightarrow VM\ VV \\
&\quad VP \rightarrow VV\ NP \\
&\quad VV \rightarrow can \mid fish \\
&\quad VM \rightarrow can \\
&\quad NP \rightarrow they \mid fish\ \}
\end{aligned}
$$

String is in the language when the cell $[0, 3]$ contains S

The CKY algorithm **recognises** strings in a CFL

0 they 1 can 2 fish 3



Toy CNF grammar:

$$
\begin{aligned}
\mathcal{N} &= \{S, NP, VP, VV, VM\} \\
\Sigma &= \{can, fish, they\} \\
\mathcal{S} &= S \\
\mathcal{P} &= \{S \rightarrow NP\ VP \\
& \quad VP \rightarrow VM\ VV \\
& \quad VP \rightarrow VV\ NP \\
& \quad VV \rightarrow can \mid fish \\
& \quad VM \rightarrow can \\
& \quad NP \rightarrow they \mid fish\ \}
\end{aligned}
$$

String is in the language when the cell $[0, 3]$ contains S

The CKY algorithm **recognises** strings in a CFL

In the general case for $A, B, C \in \mathcal{N}$ and $a \in \Sigma$:

- If $a \in \Sigma$ exists between indexes m and $m+1$, and $A \rightarrow a$ then cell $[m, m+1]$ contains A
- if cell $[i, k]$ contains B and cell $[k, j]$ contains C and $A \rightarrow BC$ then cell $[i, j]$ contains A
- String of length n is in the language when the cell $[0, n]$ contains S

The CKY algorithm only recognises a string, in order to obtain the **parse tree** we need to:

- pair each non-terminal in a cell with a 2-tuple of the cells that derived it
- allow the same non-terminal to exist more than once in any particular cell (or allow it to be paired with a list of 2-tuples)

The CKY algorithm **recognises** strings in a CFL

In the general case for $A, B, C \in \mathcal{N}$ and $a \in \Sigma$:

- If $a \in \Sigma$ exists between indexes m and $m+1$, and $A \rightarrow a$ then cell $[m, m+1]$ contains A
- if cell $[i, k]$ contains B and cell $[k, j]$ contains C and $A \rightarrow BC$ then cell $[i, j]$ contains A
- String of length n is in the language when the cell $[0, n]$ contains S

The CKY algorithm only recognises a string, in order to obtain the **parse tree** we need to:

- pair each non-terminal in a cell with a 2-tuple of the cells that derived it
- allow the same non-terminal to exist more than once in any particular cell (or allow it to be paired with a list of 2-tuples)

The CKY algorithm can be used to create a **parse**

			TO	
		1	2	3
	0			
FROM	1			
	2			
		they	*can*	*fish*

The CKY algorithm can be used to create a **parse**

		TO		
		1	2	3
	0	$NP_{(they)}$		
FROM	1			
	2			
		they	*can*	*fish*

The CKY algorithm can be used to create a **parse**

		TO		
		1	2	3
	0	$NP_{(they)}$		
FROM	1		$VV_{(can)}$ $VM_{(can)}$	
	2			
		they	*can*	*fish*

The CKY algorithm can be used to create a **parse**

		TO 1	2	3
	0	$NP_{(they)}$	.	
FROM	1		$VV_{(can)}$ $VM_{(can)}$	
	2			
		they	*can*	*fish*

The CKY algorithm can be used to create a **parse**

		TO 1	2	3
	0	$NP_{(they)}$	.	
FROM	1		$VV_{(can)}$ $VM_{(can)}$	
	2			$VV_{(fish)}$ $NP_{(fish)}$
		they	*can*	*fish*

The CKY algorithm can be used to create a **parse**

TO

		1	2	3
	0	$NP_{(they)}$	.	
FROM	1		$VV_{(can)}$ $VM_{(can)}$	$VP_{1 \rightarrow ([1,2]_{VV},[2,3]_{NP})}$ $VP_{2 \rightarrow ([1,2]_{VM},[2,3]_{VV})}$
	2			$VV_{(fish)}$ $NP_{(fish)}$
		they	*can*	*fish*

The CKY algorithm can be used to create a **parse**

		TO 1	2	3
FROM	0	$NP_{(they)}$	·	$S_{1 \rightarrow ([0,1]_{NP},[1,3]_{VP_1})}$ $S_{2 \rightarrow ([0,1]_{NP},[1,3]_{VP_2})}$
	1		$VV_{(can)}$ $VM_{(can)}$	$VP_{1 \rightarrow ([1,2]_{VV},[2,3]_{NP})}$ $VP_{2 \rightarrow ([1,2]_{VM},[2,3]_{VV})}$
	2			$VV_{(fish)}$ $NP_{(fish)}$
		they	*can*	*fish*

Ambiguous grammars derive a **parse forest**

Number of binary trees is proportional to the Catalan number

$$\texttt{Num of trees for sentence length n} = \prod_{k=2}^{n-1} \frac{(n-1)+k}{k}$$

sentence length	number of trees	sentence length	number of trees
3	2	8	429
4	5	9	1430
5	14	10	4862
6	42	11	16796
7	132	12	58786

We need parsing algorithms that can efficiently store the parse forest and not derive shared parts of tree more than once—

Ambiguous grammars derive a **parse forest**

Number of binary trees is proportional to the Catalan number

$$\texttt{Num of trees for sentence length n} = \prod_{k=2}^{n-1} \frac{(n-1)+k}{k}$$

sentence length	number of trees	sentence length	number of trees
3	2	8	429
4	5	9	1430
5	14	10	4862
6	42	11	16796
7	132	12	58786

We need parsing algorithms that can efficiently store the parse forest and not derive shared parts of tree more than once—

Ambiguous grammars derive a **parse forest**

Number of binary trees is proportional to the Catalan number

$$\texttt{Num of trees for sentence length n} = \prod_{k=2}^{n-1} \frac{(n-1)+k}{k}$$

sentence length	number of trees	sentence length	number of trees
3	2	8	429
4	5	9	1430
5	14	10	4862
6	42	11	16796
7	132	12	58786

We need parsing algorithms that can efficiently store the parse forest and not derive shared parts of tree more than once—use packing and/or a beam (the latter requires knowledge of the probability of derivations)

Parse probabilities may be derived using a **PCFG**

- $G_{pcfg} = (\Sigma, \mathcal{N}, S, \mathcal{P}, q)$ where q is a mapping from rules in $\mathcal{P}$ to a probability and $\sum_{A \to \alpha \in \mathcal{P}} q(A \to \alpha) = 1$
- G_{pcfg} is **consistent** if the sum of all probabilities of all derivable strings equals 1 (grammars with infinite loops like $S \to S$ are inconsistent)
- The probability of a particular parse is the **product** of the probabilities of the rules that defined the parse tree. For a string W with parse tree T derived from rules $A_i \to B_i$, $i = 1...n$

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

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

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

Parse probabilities may be derived using a **PCFG**

- The probability of an ambiguous string is the sum of all the parse trees that **yield** that string

$$P(W) = \sum_{\textit{trees that yield } W} P(T, W) = \sum_{\textit{trees that yield } W} P(T)$$

- We can disambiguate multiple parses by choosing the most probable parse tree for the string

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

but

$$P(T|W) = \frac{P(T,W)}{P(W)} \rightarrow P(T, W) = P(T)$$

so

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

Rule probabilities may be estimated from **treebanks**

- A **treebank** is a corpus of parsed sentences
- Rule probabilities can be estimated from counts in a treebank:
  $P(A \rightarrow B) = P(A \rightarrow B|A) = \frac{count(A \rightarrow B)}{\sum_{\gamma} count(A \rightarrow \gamma)} = \frac{count(A \rightarrow B)}{count(A)}$
- **inside-outside algorithm** can be used when no tree bank exists

... more in later lectures

Problems with PCFGs:

- Independence ignores structural dependency within the tree
- Structure is dependent on lexical items

... more in later lectures

Rule probabilities may be estimated from **treebanks**

- A **treebank** is a corpus of parsed sentences
- Rule probabilities can be estimated from counts in a treebank:
  $$P(A \rightarrow B) = P(A \rightarrow B|A) = \frac{count(A \rightarrow B)}{\sum_{\gamma} count(A \rightarrow \gamma)} = \frac{count(A \rightarrow B)}{count(A)}$$
- **inside-outside algorithm** can be used when no tree bank exists

... more in later lectures

Problems with PCFGs:

- Independence ignores structural dependency within the tree
- Structure is dependent on lexical items

... more in later lectures

Rule probabilities may be estimated from **treebanks**

- A **treebank** is a corpus of parsed sentences
- Rule probabilities can be estimated from counts in a treebank:
  $P(A \rightarrow B) = P(A \rightarrow B|A) = \frac{count(A \rightarrow B)}{\sum_{\gamma} count(A \rightarrow \gamma)} = \frac{count(A \rightarrow B)}{count(A)}$
- **inside-outside algorithm** can be used when no tree bank exists

... more in later lectures

Problems with PCFGs:

- Independence ignores structural dependency within the tree
- Structure is dependent on lexical items

... more in later lectures

Rule probabilities may be estimated from **treebanks**

- A **treebank** is a corpus of parsed sentences
- Rule probabilities can be estimated from counts in a treebank:
  $P(A \rightarrow B) = P(A \rightarrow B|A) = \frac{count(A \rightarrow B)}{\sum_{\gamma} count(A \rightarrow \gamma)} = \frac{count(A \rightarrow B)}{count(A)}$
- **inside-outside algorithm** can be used when no tree bank exists

... more in later lectures

Problems with PCFGs:

- Independence ignores structural dependency within the tree
- Structure is dependent on lexical items

... more in later lectures

Probabilistic CFGs may be incorporated into CKY

	1	2	3
0			
1			
2			
	they	*can*	*fish*

$$
\begin{aligned}
\mathcal{N} &= \{S, NP, VP, VV, VM\} \\
\Sigma &= \{can, fish, they\} \\
S &= S \\
\mathcal{P} &= \{S \rightarrow NP\ VP\ 1.0 \\
& \quad VP \rightarrow VM\ VV\ 0.9 \\
& \quad VP \rightarrow VV\ NP\ 0.1 \\
& \quad VV \rightarrow can\ 0.2 \mid fish\ 0.8 \\
& \quad VM \rightarrow can\ 1.0 \\
& \quad NP \rightarrow they\ 0.5 \mid fish\ 0.5
\end{aligned}
$$

- For the best parse keep most probable non-terminal at each node
- Otherwise can pack and operate a beam

Probabilistic CFGs may be incorporated into CKY

	1	2	3
0	$NP^{0.5}_{(they)}$		
1			
2			
	they	*can*	*fish*

$$
\begin{aligned}
\mathcal{N} &= \{S, NP, VP, VV, VM\} \\
\Sigma &= \{can, fish, they\} \\
S &= S \\
\mathcal{P} &= \{S \rightarrow NP\ VP\ 1.0 \\
&\quad VP \rightarrow VM\ VV\ 0.9 \\
&\quad VP \rightarrow VV\ NP\ 0.1 \\
&\quad VV \rightarrow can\ 0.2 \mid fish\ 0.8 \\
&\quad VM \rightarrow can\ 1.0 \\
&\quad NP \rightarrow they\ 0.5 \mid fish\ 0.5
\end{aligned}
$$

- For the best parse keep most probable non-terminal at each node
- Otherwise can pack and operate a beam

Probabilistic CFGs may be incorporated into CKY

	1	2	3
0	$NP^{0.5}_{(they)}$		
1		$VV^{0.2}_{(can)}$ $VM^{1.0}_{(can)}$	
2			
	they	*can*	*fish*

$$
\begin{aligned}
\mathcal{N} &= \{S, NP, VP, VV, VM\} \\
\Sigma &= \{can, fish, they\} \\
S &= S \\
\mathcal{P} &= \{S \rightarrow NP\ VP\ 1.0 \\
&\quad VP \rightarrow VM\ VV\ 0.9 \\
&\quad VP \rightarrow VV\ NP\ 0.1 \\
&\quad VV \rightarrow can\ 0.2 \mid fish\ 0.8 \\
&\quad VM \rightarrow can\ 1.0 \\
&\quad NP \rightarrow they\ 0.5 \mid fish\ 0.5
\end{aligned}
$$

- For the best parse keep most probable non-terminal at each node
- Otherwise can pack and operate a beam

Probabilistic CFGs may be incorporated into CKY

	1	2	3
0	$NP^{0.5}_{(they)}$	.	
1		$VV^{0.2}_{(can)}$ $VM^{1.0}_{(can)}$	
2			
	they	*can*	*fish*

$$
\begin{aligned}
\mathcal{N} &= \{S, NP, VP, VV, VM\} \\
\Sigma &= \{can, fish, they\} \\
S &= S \\
\mathcal{P} &= \{S \rightarrow NP\ VP\ 1.0 \\
& \quad VP \rightarrow VM\ VV\ 0.9 \\
& \quad VP \rightarrow VV\ NP\ 0.1 \\
& \quad VV \rightarrow can\ 0.2 \mid fish\ 0.8 \\
& \quad VM \rightarrow can\ 1.0 \\
& \quad NP \rightarrow they\ 0.5 \mid fish\ 0.5
\end{aligned}
$$

- For the best parse keep most probable non-terminal at each node
- Otherwise can pack and operate a beam

Probabilistic CFGs may be incorporated into CKY

	1	2	3
0	$NP^{0.5}_{(they)}$	.	
1		$VV^{0.2}_{(can)}$ $VM^{1.0}_{(can)}$	
2			$VV^{0.8}_{(fish)}$ $NP^{0.5}_{(fish)}$
	they	*can*	*fish*

$$\begin{aligned}
\mathcal{N} &= \{S, NP, VP, VV, VM\} \\
\Sigma &= \{can, fish, they\} \\
S &= S \\
\mathcal{P} &= \{S \rightarrow NP\ VP\ 1.0 \\
&\quad VP \rightarrow VM\ VV\ 0.9 \\
&\quad VP \rightarrow VV\ NP\ 0.1 \\
&\quad VV \rightarrow can\ 0.2 \mid fish\ 0.8 \\
&\quad VM \rightarrow can\ 1.0 \\
&\quad NP \rightarrow they\ 0.5 \mid fish\ 0.5
\end{aligned}$$

- For the best parse keep most probable non-terminal at each node
- Otherwise can pack and operate a beam

Probabilistic CFGs may be incorporated into CKY

	1	2	3
0	$NP^{0.5}_{(they)}$	.	
1		$VV^{0.2}_{(can)}$ $VM^{1.0}_{(can)}$	$VP^{0.2*0.5*0.1=0.01}_{1\rightarrow([1,2]_{VV},[2,3]_{NP})}$ $VP^{1.0*0.8*0.9=0.72}_{2\rightarrow([1,2]_{VM},[2,3]_{VV})}$
2			$VV^{0.8}_{(fish)}$ $NP^{0.5}_{(fish)}$
	they	*can*	*fish*

$$
\begin{aligned}
\mathcal{N} &= \{S, NP, VP, VV, VM\} \\
\Sigma &= \{can, fish, they\} \\
S &= S \\
\mathcal{P} &= \{S \rightarrow NP\ VP\ 1.0 \\
&\quad VP \rightarrow VM\ VV\ 0.9 \\
&\quad VP \rightarrow VV\ NP\ 0.1 \\
&\quad VV \rightarrow can\ 0.2 \mid fish\ 0.8 \\
&\quad VM \rightarrow can\ 1.0 \\
&\quad NP \rightarrow they\ 0.5 \mid fish\ 0.5
\end{aligned}
$$

- For the best parse keep most probable non-terminal at each node
- Otherwise can pack and operate a beam

Probabilistic CFGs may be incorporated into CKY

	1	2	3
0	$NP^{0.5}_{(they)}$	.	$S^{0.5*1.0*0.8*0.9*1.0=0.36}_{([0,1]_{NP},[1,3]_{VP})}$
1		$VV^{0.2}_{(can)}$ $VM^{1.0}_{(can)}$	$VP^{1.0*0.8*0.9=0.72}_{([1,2]_{VM},[2,3]_{VV})}$
2			$VV^{0.8}_{(fish)}$ $NP^{0.5}_{(fish)}$
	they	*can*	*fish*

$$
\begin{aligned}
\mathcal{N} &= \{S, NP, VP, VV, VM\} \\
\Sigma &= \{can, fish, they\} \\
S &= S \\
\mathcal{P} &= \{S \rightarrow NP\ VP\ 1.0 \\
& \quad VP \rightarrow VM\ VV\ 0.9 \\
& \quad VP \rightarrow VV\ NP\ 0.1 \\
& \quad VV \rightarrow can\ 0.2 \mid fish\ 0.8 \\
& \quad VM \rightarrow can\ 1.0 \\
& \quad NP \rightarrow they\ 0.5 \mid fish\ 0.5
\end{aligned}
$$

- For the best parse keep most probable non-terminal at each node
- Otherwise can pack and operate a beam