Introduction to Music Processing

Hierarchical Models

Harmonies and Context-Free Grammars

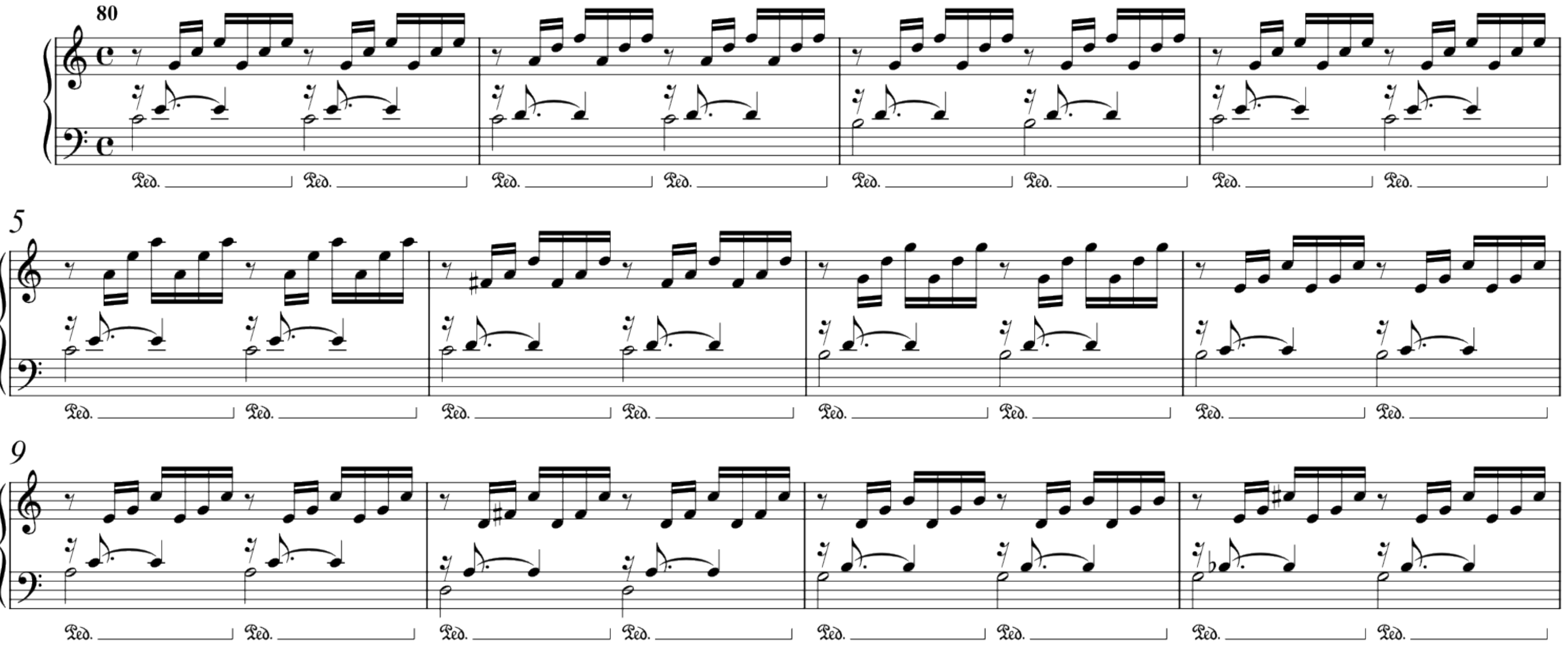
Dr Robert Lieck
robert.lieck@durham.ac.uk

Music as a Hierarchy

Prélude No. 1 in C Major

from "Das Wohltemperierte Klavier" Book I
BWV 846

Johann Sebastian Bach
(1685 - 1750)

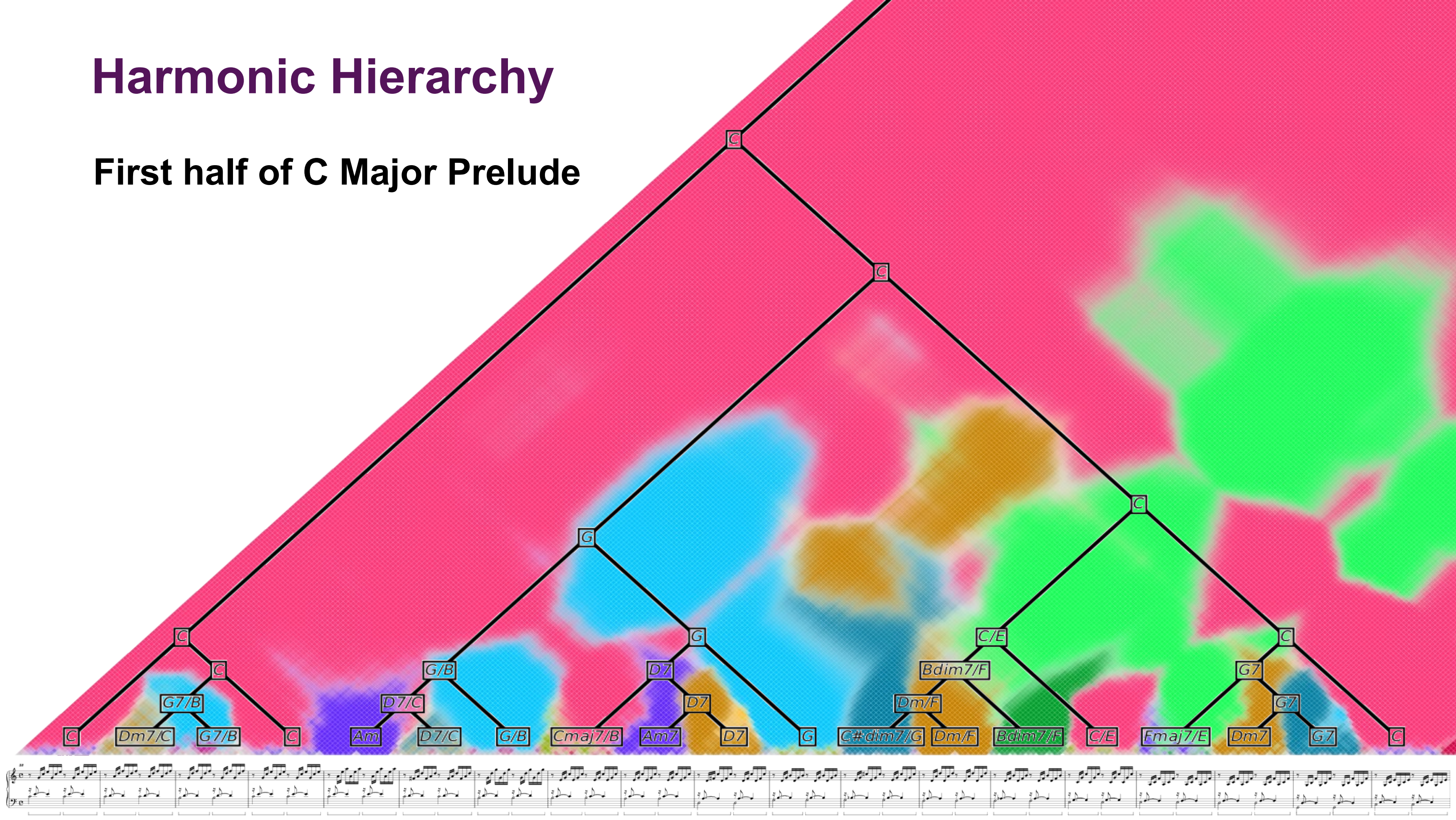


Durham University

https://www.youtube.com/watch?v=frxT2qB1POQ

Harmonic Hierarchy

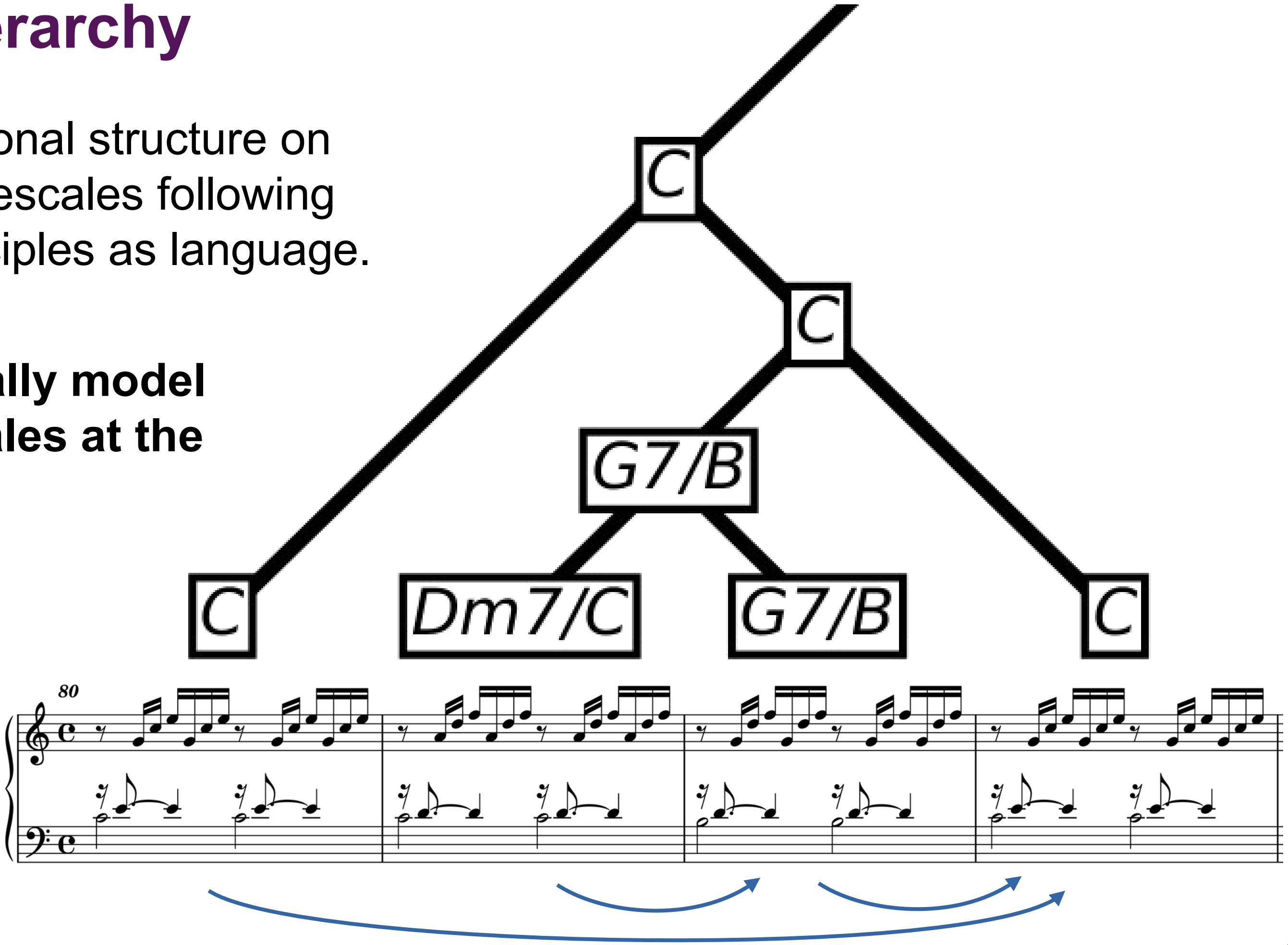
First half of C Major Prelude



Harmonic Hierarchy

Music has tonal structure on multiple timescales following similar principles as language.

Hierarchically model all time scales at the same time!



Context-Free Grammars (CFGs)

A context-free grammar consists of:

1) A set of latent non-terminal symbols **X**

2) A set of observed terminal symbols **Y**

3) A set of rules **R: X ⟶ (X∪Y)***

   typically only (Chomsky Normal Form) **X ⟶ X X or X ⟶ Y**

4) A (non-terminal) start symbol **S ∈ X**

5) *Optional:* Probabilities **P** (or weights) for the rules

Rhythm Grammar

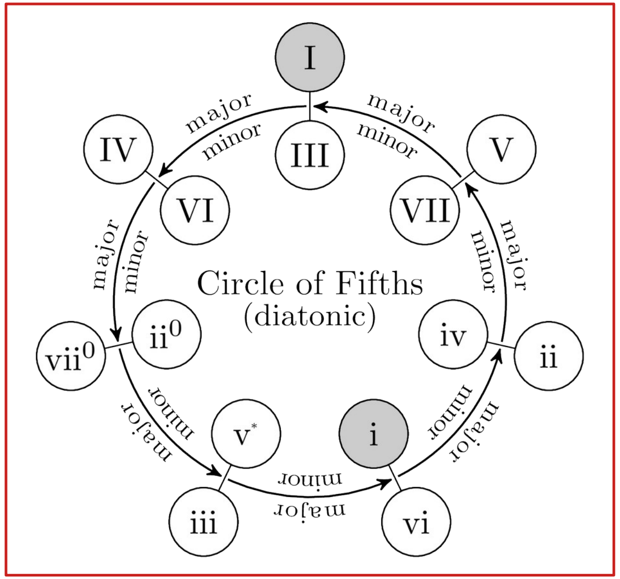
- X = {T}
- Y = {b, r}
- R & P:

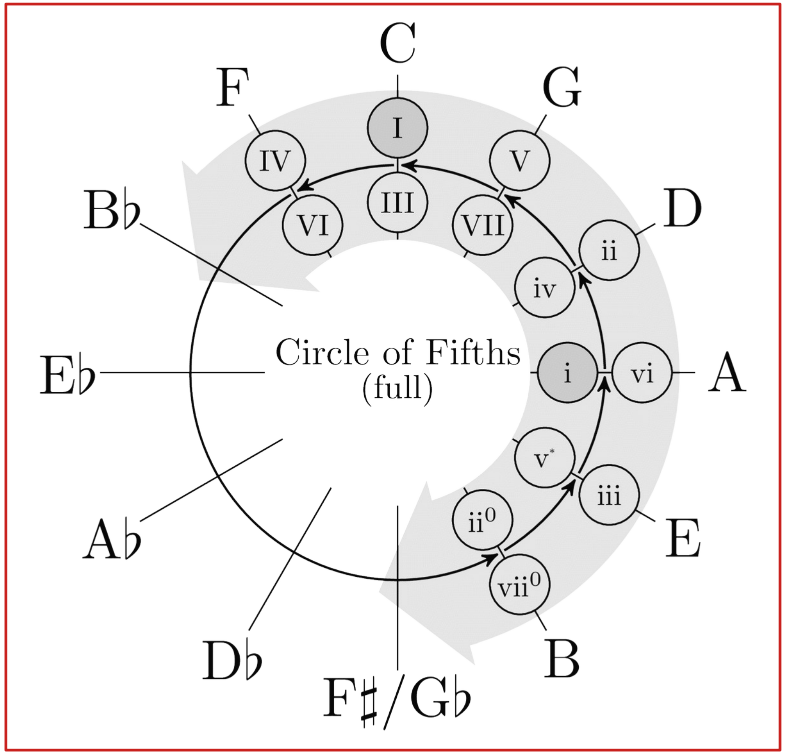
$$\begin{aligned} \text{split } (p_s)&: \quad T \longrightarrow TT \\ \text{beat } (p_b)&: \quad T \longrightarrow b \\ \text{rest } (p_r)&: \quad T \longrightarrow r \end{aligned}$$

Context-Free Grammars: Harmony Grammars

Falling Fifths

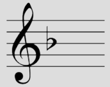
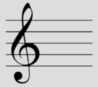
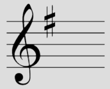
- Fundamental in Western harmony
- Goal-directed "relaxation" (e.g. in cadences)

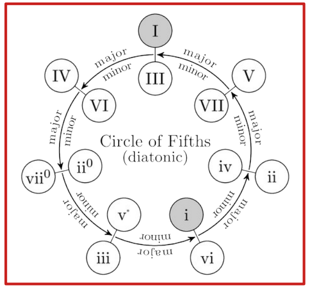


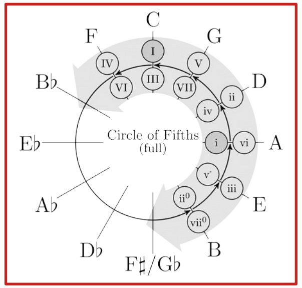


Simple: https://youtu.be/pRl_PqEHxSk?si=Xd2b86uCoDlhq6Ry
Advanced: https://youtu.be/WcpR5TpG7Ig?si=3mfkSVEPyzNSTqtB

Context-Free Grammars: Harmony Grammars

Non-Terminals X							
Major Key	I	IV	viiº	iii	vi	ii	V
Minor Key	III	VI	iiº	v*	i	iv	VII
Terminals Y							
…	…	…	…	…	…	…	…
	F	B♭	Eº	Am	Dm	Gm	C
	**C**	**F**	**Bº**	**Em**	**Am**	**Dm**	**G**
	G	C	F♯º	Bm	Em	Am	D
…	…	…	…	…	…	…	…





Non-Terminal Rules

Major Key	**Minor Key**
I ⟶ V I	III ⟶ VII III
IV ⟶ I IV	VI ⟶ III VI
viiº ⟶ IV viiº	iiº ⟶ VI iiº
iii ⟶ viiº iii	v* ⟶ iiº v*
vi ⟶ iii vi	i ⟶ v* i
ii ⟶ vi ii	iv ⟶ i iv
V ⟶ ii V	VII ⟶ iv VII
I ⟶ I I	i ⟶ i i

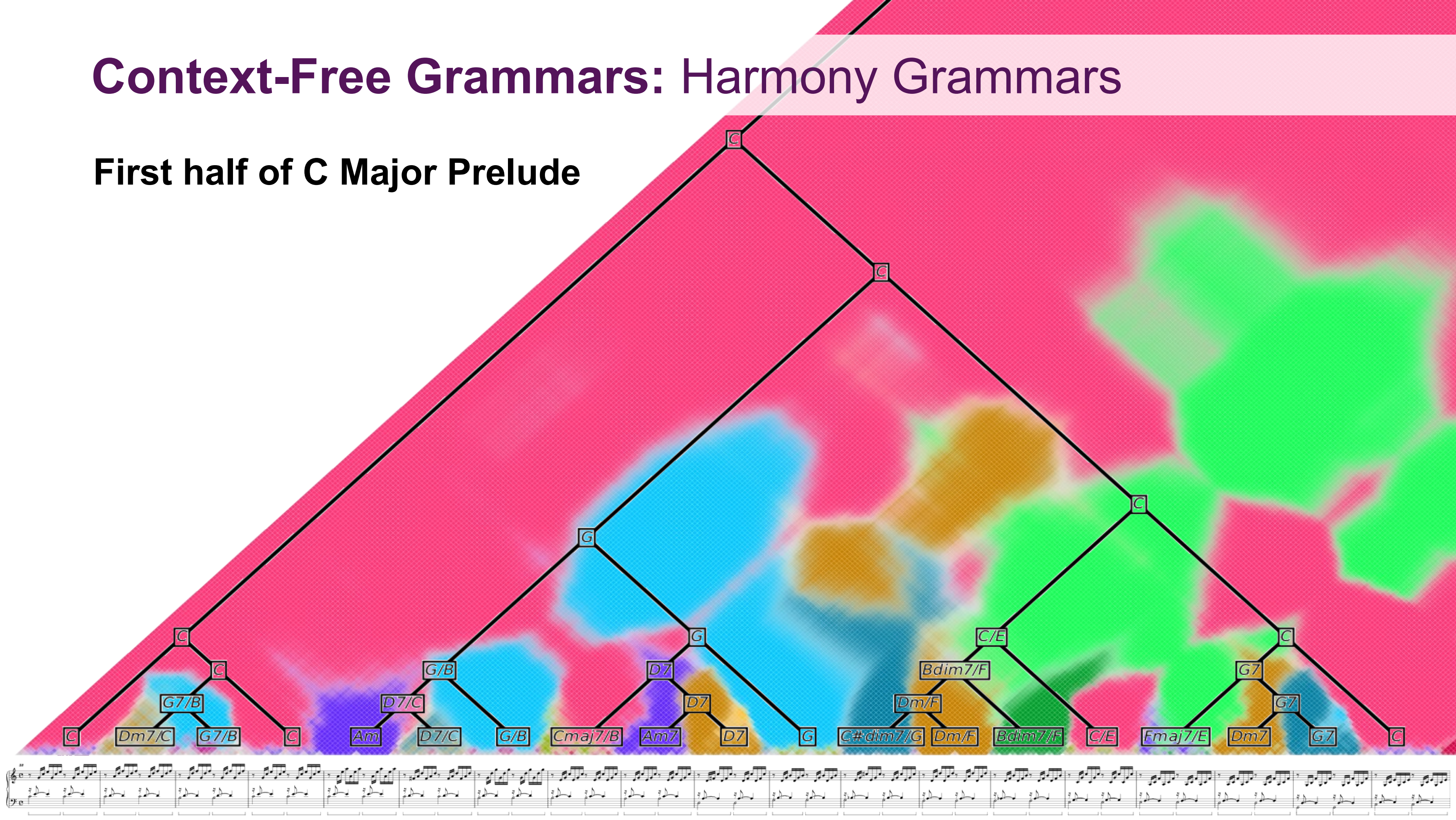
Terminal Rules

C Major	**A Minor**
I ⟶ C	III ⟶ C
IV ⟶ F	VI ⟶ F
viiº ⟶ Bº	iiº ⟶ Bº
iii ⟶ Em	v* ⟶ Em
vi ⟶ Am	i ⟶ Am
ii ⟶ Dm	iv ⟶ Dm
V ⟶ G	VII ⟶ G

Durham University

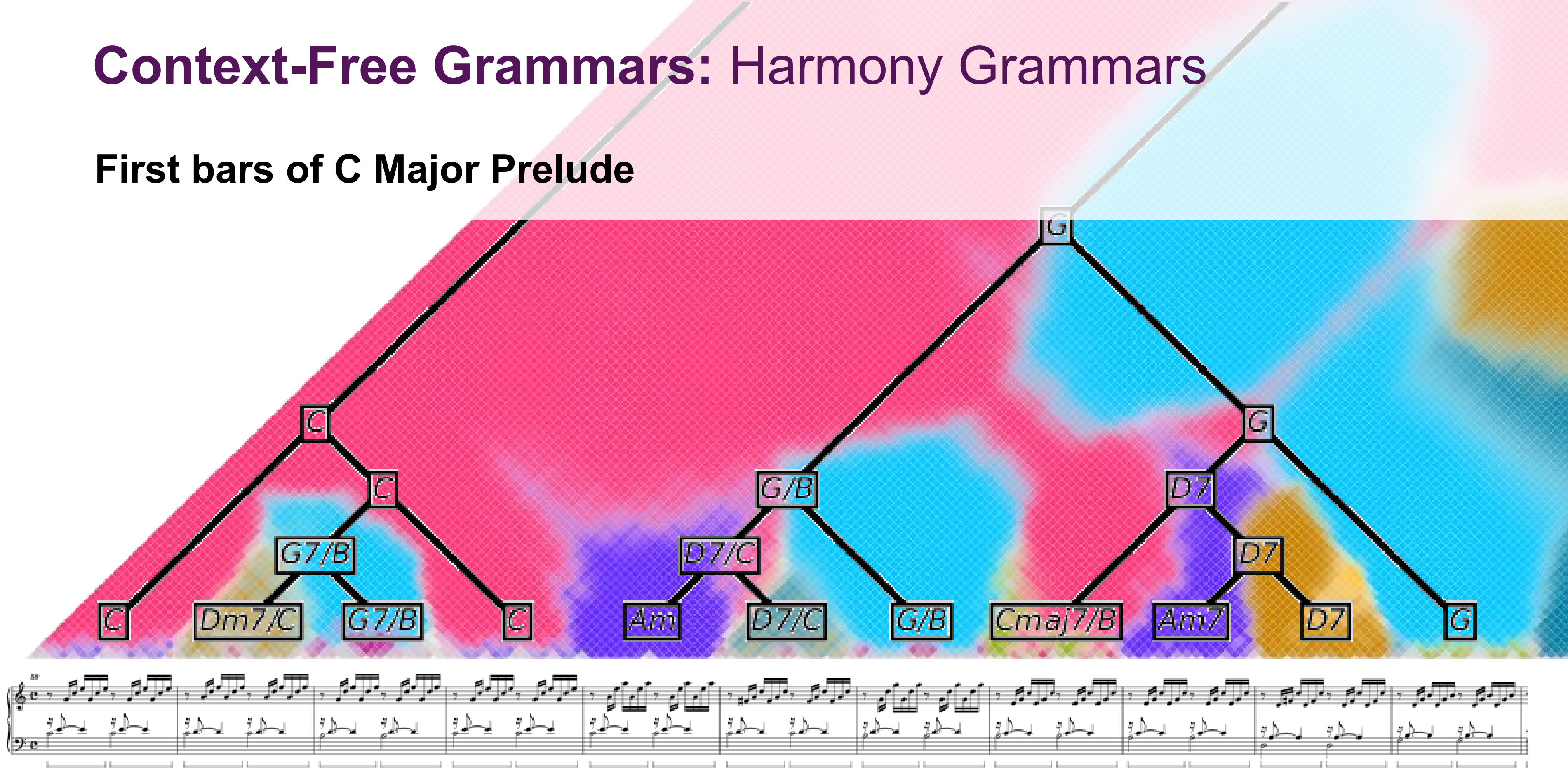
Context-Free Grammars: Harmony Grammars

First half of C Major Prelude



Context-Free Grammars: Harmony Grammars

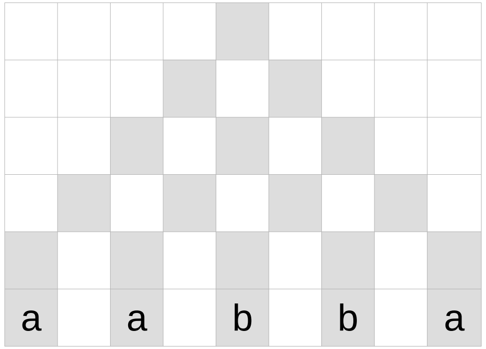
First bars of C Major Prelude



Context-Free Grammars: Analysis (Parsing)

Generation versus Analysis

- If we are given the rules, we can easily generate by applying them recursively.
- How can we infer (“reverse engineer”) the generation process?
- Work your way up in reverse order from the bottom to the top:
  - Reuse sub-solutions from further down.
  - Make sure you always cover the whole span by splitting it at a specific point.
  - If start symbol is at the top, the sequence is valid.
  - Reconstruct possible generation trees top-down.



Non-Terminals
{A, B}

Terminals
{a, b}

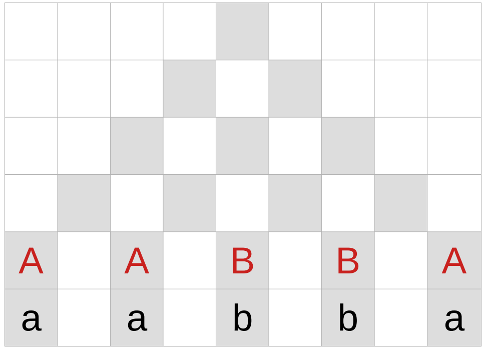
Start Symbol
S = A

Rules
$A \longrightarrow A\ A$
$A \longrightarrow B\ A$
$A \longrightarrow a$
$B \longrightarrow b$

Context-Free Grammars: Analysis (Parsing)

Generation versus Analysis

- If we are given the rules, we can easily generate by applying them recursively.
- How can we infer ("reverse engineer") the generation process?
- Work your way up in reverse order from the bottom to the top:
  - Reuse sub-solutions from further down.
  - Make sure you always cover the whole span by splitting it at a specific point.
  - If start symbol is at the top, the sequence is valid.
  - Reconstruct possible generation trees top-down.

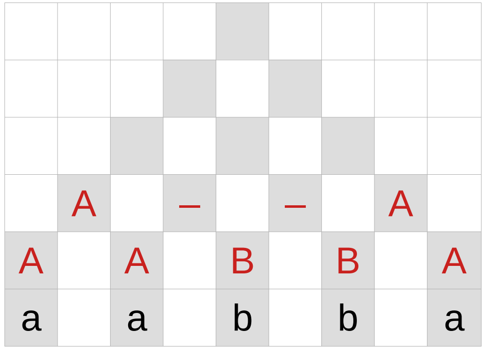


Non-Terminals	**Rules**
{A, B}	$A \longrightarrow A\,A$
	$A \longrightarrow B\,A$
Terminals	$A \longrightarrow a$
{a, b}	$B \longrightarrow b$
Start Symbol	
S = A	

Context-Free Grammars: Analysis (Parsing)

Generation versus Analysis

- If we are given the rules, we can easily generate by applying them recursively.
- How can we infer (“reverse engineer”) the generation process?
- Work your way up in reverse order from the bottom to the top:
  - Reuse sub-solutions from further down.
  - Make sure you always cover the whole span by splitting it at a specific point.
  - If start symbol is at the top, the sequence is valid.
  - Reconstruct possible generation trees top-down.

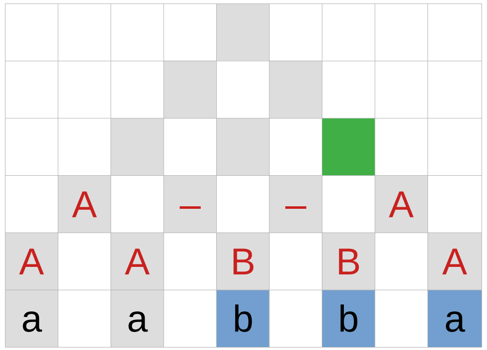


Non-Terminals	**Rules**
{A, B}	$A \longrightarrow A\ A$
	$A \longrightarrow B\ A$
Terminals	$A \longrightarrow a$
{a, b}	$B \longrightarrow b$
Start Symbol	
S = A	

Context-Free Grammars: Analysis (Parsing)

Generation versus Analysis

- If we are given the rules, we can easily generate by applying them recursively.
- How can we infer (“reverse engineer”) the generation process?
- Work your way up in reverse order from the bottom to the top:
  - Reuse sub-solutions from further down.
  - Make sure you always cover the whole span by splitting it at a specific point.
  - If start symbol is at the top, the sequence is valid.
  - Reconstruct possible generation trees top-down.

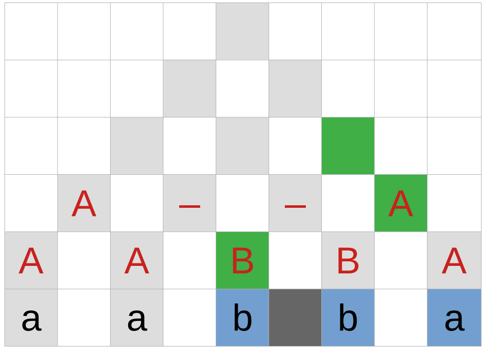


Non-Terminals	**Rules**
{A, B}	$A \longrightarrow A\ A$
	$A \longrightarrow B\ A$
Terminals	$A \longrightarrow a$
{a, b}	$B \longrightarrow b$
Start Symbol	
S = A	

Context-Free Grammars: Analysis (Parsing)

Generation versus Analysis

- If we are given the rules, we can easily generate by applying them recursively.
- How can we infer (“reverse engineer”) the generation process?
- Work your way up in reverse order from the bottom to the top:
  - Reuse sub-solutions from further down.
  - Make sure you always cover the whole span by splitting it at a specific point.
  - If start symbol is at the top, the sequence is valid.
  - Reconstruct possible generation trees top-down.



Non-Terminals
{A, B}

Terminals
{a, b}

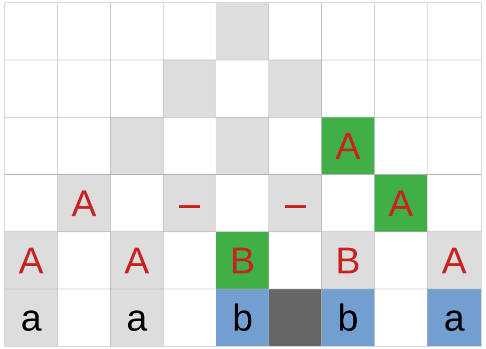
Start Symbol
S = A

Rules
$A \longrightarrow A\ A$
$A \longrightarrow B\ A$
$A \longrightarrow a$
$B \longrightarrow b$

Context-Free Grammars: Analysis (Parsing)

Generation versus Analysis

- If we are given the rules, we can easily generate by applying them recursively.
- How can we infer (“reverse engineer”) the generation process?
- Work your way up in reverse order from the bottom to the top:
  - Reuse sub-solutions from further down.
  - Make sure you always cover the whole span by splitting it at a specific point.
  - If start symbol is at the top, the sequence is valid.
  - Reconstruct possible generation trees top-down.

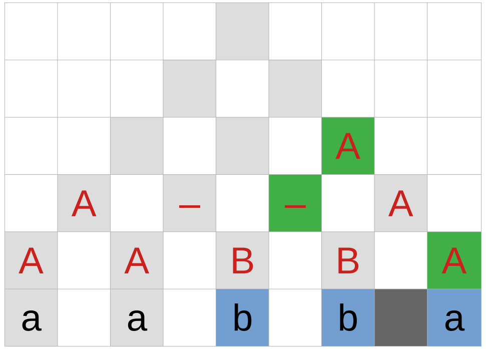


Non-Terminals	**Rules**
{A, B}	A ⟶ A A
	A ⟶ B A
Terminals	A ⟶ a
{a, b}	B ⟶ b
Start Symbol	
S = A	

Context-Free Grammars: Analysis (Parsing)

Generation versus Analysis

- If we are given the rules, we can easily generate by applying them recursively.
- How can we infer (“reverse engineer”) the generation process?
- Work your way up in reverse order from the bottom to the top:
  - Reuse sub-solutions from further down.
  - Make sure you always cover the whole span by splitting it at a specific point.
  - If start symbol is at the top, the sequence is valid.
  - Reconstruct possible generation trees top-down.

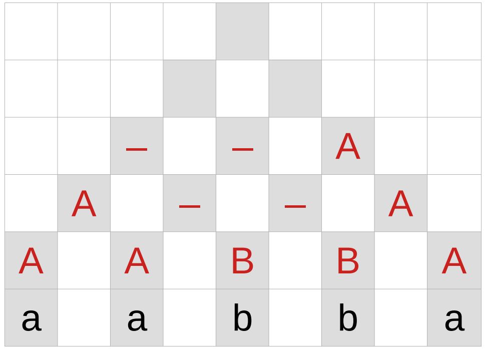


Non-Terminals	**Rules**
{A, B}	$A \longrightarrow A\ A$
	$A \longrightarrow B\ A$
Terminals	$A \longrightarrow a$
{a, b}	$B \longrightarrow b$
Start Symbol	
S = A	

Context-Free Grammars: Analysis (Parsing)

Generation versus Analysis

- If we are given the rules, we can easily generate by applying them recursively.
- How can we infer (“reverse engineer”) the generation process?
- Work your way up in reverse order from the bottom to the top:
  - Reuse sub-solutions from further down.
  - Make sure you always cover the whole span by splitting it at a specific point.
  - If start symbol is at the top, the sequence is valid.
  - Reconstruct possible generation trees top-down.



		–		–		A		
	A		–		–		A	
A		A		B		B		A
a		a		b		b		a

Non-Terminals	**Rules**
{A, B}	$A \longrightarrow A\ A$
	$A \longrightarrow B\ A$
Terminals	$A \longrightarrow a$
{a, b}	$B \longrightarrow b$
Start Symbol	
S = A	

Context-Free Grammars: Analysis (Parsing)

Generation versus Analysis

- If we are given the rules, we can easily generate by applying them recursively.
- How can we infer (“reverse engineer”) the generation process?
- Work your way up in reverse order from the bottom to the top:
  - Reuse sub-solutions from further down.
  - Make sure you always cover the whole span by splitting it at a specific point.
  - If start symbol is at the top, the sequence is valid.
  - Reconstruct possible generation trees top-down.

			–		A			
		–		–		A		
	A		–		–		A	
A		A		B		B		A
a		a		b		b		a

Non-Terminals	**Rules**
{A, B}	$A \longrightarrow A\ A$
	$A \longrightarrow B\ A$
Terminals	$A \longrightarrow a$
{a, b}	$B \longrightarrow b$
Start Symbol	
S = A	

Context-Free Grammars: Analysis (Parsing)

Generation versus Analysis

- If we are given the rules, we can easily generate by applying them recursively.
- How can we infer (“reverse engineer”) the generation process?
- Work your way up in reverse order from the bottom to the top:
  - Reuse sub-solutions from further down.
  - Make sure you always cover the whole span by splitting it at a specific point.
  - If start symbol is at the top, the sequence is valid.
  - Reconstruct possible generation trees top-down.

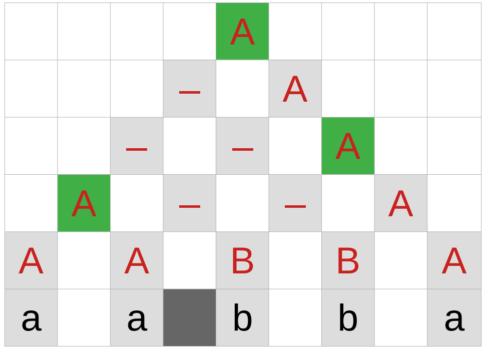
				A				
			–		A			
		–		–		A		
	A		–		–		A	
A		A		B		B		A
a	■	a		b		b		a

Non-Terminals	**Rules**
{A, B}	$A \longrightarrow A\ A$
	$A \longrightarrow B\ A$
Terminals	$A \longrightarrow a$
{a, b}	$B \longrightarrow b$
Start Symbol	
S = A	

Context-Free Grammars: Analysis (Parsing)

Generation versus Analysis

- If we are given the rules, we can easily generate by applying them recursively.
- How can we infer (“reverse engineer”) the generation process?
- Work your way up in reverse order from the bottom to the top:
  - Reuse sub-solutions from further down.
  - Make sure you always cover the whole span by splitting it at a specific point.
  - If start symbol is at the top, the sequence is valid.
  - Reconstruct possible generation trees top-down.



				A				
			–		A			
		–		–		A		
	A		–		–		A	
A		A		B		B		A
a		a	■	b		b		a

Non-Terminals	**Rules**
{A, B}	$A \longrightarrow A\ A$
	$A \longrightarrow B\ A$
Terminals	$A \longrightarrow a$
{a, b}	$B \longrightarrow b$
Start Symbol	
S = A	

Context-Free Grammars: Analysis (Parsing)

Generation versus Analysis

- If we are given the rules, we can easily generate by applying them recursively.
- How can we infer (“reverse engineer”) the generation process?
- Work your way up in reverse order from the bottom to the top:
  - Reuse sub-solutions from further down.
  - Make sure you always cover the whole span by splitting it at a specific point.
  - If start symbol is at the top, the sequence is valid.
  - Reconstruct possible generation trees top-down.

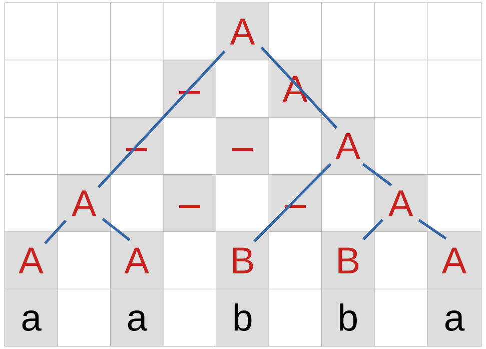
				A				
			–		A			
		–		–		A		
	A		–		–		A	
A		A		B		B		A
a		a		b		b		a

Non-Terminals	Rules
{A, B}	$A \longrightarrow A\ A$
	$A \longrightarrow B\ A$
Terminals	$A \longrightarrow a$
{a, b}	$B \longrightarrow b$
Start Symbol	
S = A	

Context-Free Grammars: Analysis (Parsing)

Generation versus Analysis

- If we are given the rules, we can easily generate by applying them recursively.
- How can we infer ("reverse engineer") the generation process?
- Work your way up in reverse order from the bottom to the top:
  - Reuse sub-solutions from further down.
  - Make sure you always cover the whole span by splitting it at a specific point.
  - If start symbol is at the top, the sequence is valid.
  - Reconstruct possible generation trees top-down.

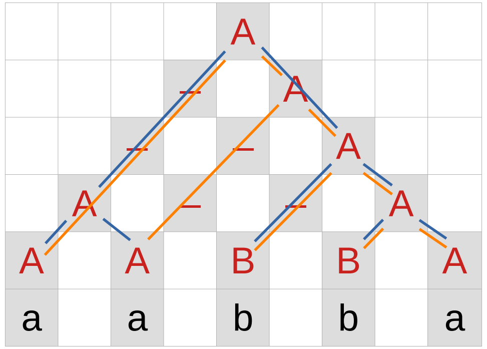


Non-Terminals	**Rules**
{A, B}	A ⟶ A A
	A ⟶ B A
Terminals	A ⟶ a
{a, b}	B ⟶ b
Start Symbol	
S = A	

Context-Free Grammars: Analysis (Parsing)

Generation versus Analysis

- If we are given the rules, we can easily generate by applying them recursively.
- How can we infer ("reverse engineer") the generation process?
- Work your way up in reverse order from the bottom to the top:
  - Reuse sub-solutions from further down.
  - Make sure you always cover the whole span by splitting it at a specific point.
  - If start symbol is at the top, the sequence is valid.
  - Reconstruct possible generation trees top-down.



Non-Terminals	**Rules**
{A, B}	A ⟶ A A
	A ⟶ B A
Terminals	A ⟶ a
{a, b}	B ⟶ b
Start Symbol	
S = A	

Context-Free Grammars: CYK Algorithm

The Cocke–Younger–Kasami (CYK) Algorithm

```
for start in {0, …, n – 1}: # bottom row
  fill_terminal_cell(start)
for level in {2, …, n}: # all other rows
  for start in {0, …, n – level}:
    end = start + level
    for split in {start + 1 ,…, start + level – 1}:
      fill_non_terminal_cell(start, split, end)
```

```
def fill_terminal_cell(start):
  for (x → y) in terminal_rules:
    if y is symbol after start:
      add x to chart[start, start + 1]
```

```
def fill_non_terminal_cell(start, split, end):
  for (x1 → x2 x3) in non_terminal_rules:
    if (x2 is in chart[start, split] and
        x3 is in chart[split, end]):
      add (split for) x1 to chart[start, end]
```

l = 5						A					
l = 4					–		A				
l = 3				–		–		A			
l = 2			A		–		–		A		
l = 1		A		A		B		B		A	
		a		a		b		b		a	
	0		1		2		3		4		5

Remembering possible splits allows reconstructing trees!

Probabilistic Context-Free Grammars (PCFGs)

Computing Probabilities

- In non-probabilistic CFGs, we compute Boolean values (True/False).
- We can use the exact same algorithm to compute probabilities instead.
  The result of parsing then is the probability of generating the given sequence from the given rules.
- In fact, we can use any semiring (defining *addition* and *multiplication*).
  Boolean semiring for CFGs; probabilities for PCFGs; lists of probabilities for top-*k* parsing

```
def fill_non_terminal_cell(start, split, end):
  for (x1 → x2 x3) in non_terminal_rules:
    if (x2 is in chart[start, split] and
        x3 is in chart[split, end]):
      add x1 to chart[start, end]
```

multiplication (and)

addition (or)

Multiplication (and)

	0	1
0	0	0
1	0	1

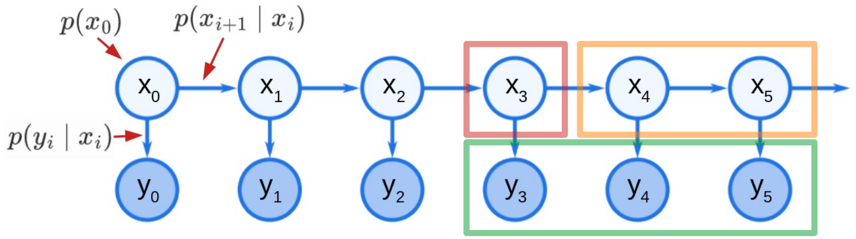
Addition (or)

	0	1
0	0	1
1	1	1

Goodman J (1999) Semiring parsing. Computational Linguistics 25:573–605

HMM-Based Chord Recognition

Forward-Backward Algorithm



~~forward~~
outside

$$\alpha(x_i) := p(y_0, \ldots, y_{i-1}, x_i)$$
$$= \sum_{x_{i-1}} p(x_i \mid x_{i-1})\, p(y_{i-1} \mid x_{i-1})\, p(y_0, \ldots, y_{i-2}, x_{i-1})$$
$$= \sum_{x_{i-1}} p(x_i \mid x_{i-1})\, p(y_{i-1} \mid x_{i-1})\, \alpha(x_{i-1})$$
$$\alpha(x_0) = p(x_0)$$

~~backward~~
inside

$$\beta(x_i) := p(y_i, y_{i+1}, \ldots, y_n \mid x_i)$$
$$= \sum_{x_{i+1}} p(x_{i+1} \mid x_i)\, p(y_i \mid x_i)\, p(y_{i+1}, \ldots, y_n \mid x_{i+1})$$
$$= \sum_{x_{i+1}} p(x_{i+1} \mid x_i)\, p(y_i \mid x_i)\, \beta(x_{i+1})$$
$$\beta(x_n) = p(y_n \mid x_n)$$

$$\alpha(x_i)\beta(x_i) = p(y_0, \ldots, y_n, x_i)$$

data likelihood

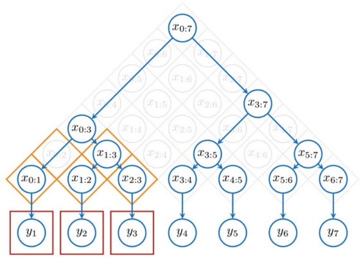
$$\sum_{x_i} \alpha(x_i)\beta(x_i) = p(y_0, \ldots, y_n) = \ell(\text{data})$$

marginals

$$\frac{\alpha(x_i)\beta(x_i)}{\ell(\text{data})} = p(x_i \mid y_0, \ldots, y_n)$$

Durham University

Recursive Bayesian Networks & Probabilistic Context-Free Grammars



outside

$$\alpha(x_{j:k}) = \left[\sum_{i=0}^{j-1} \iint p_{\mathrm{N}}(x_{i:j}, x_{j:k} \mid x_{i:k})\, \alpha(x_{i:k})\, \beta(x_{i:j})\right] + \left[\sum_{l=k+1}^{n} \iint p_{\mathrm{N}}(x_{j:k}, x_{k:l} \mid x_{j:l})\, \alpha(x_{j:l})\, \beta(x_{k:l})\right]$$

inside

$$\beta(x_{i:k}) = \sum_{j=i+1}^{k-1} \iint p_{\mathrm{N}}(x_{i:j}, x_{j:k} \mid x_{i:k})\, \beta(x_{i:j})\, \beta(x_{j:k})$$

$$\alpha(x_i)\beta(x_i) = p(y_0, \ldots, y_n, x_i)$$

data likelihood

$$\sum_{x_i} \alpha(x_i)\beta(x_i) = p(y_0, \ldots, y_n) = \ell(\text{data})$$

marginals

$$\frac{\alpha(x_i)\beta(x_i)}{\ell(\text{data})} = p(x_i \mid y_0, \ldots, y_n)$$

Durham University

References

1) Lerdahl F, Jackendoff R (1983) A generative theory of tonal music. MIT press

2) Goodman J (1999) Semiring parsing. Computational Linguistics 25:573–605

3) Huron D, Ommen A (2006) An empirical study of syncopation in American popular music, 1890–1939. Music Theory Spectrum 28:211–231

4) Grune D, Jacobs CJ (2007) Parsing techniques. Monographs in Computer Science Springer

5) Rohrmeier M (2011) Towards a generative syntax of tonal harmony. Journal of Mathematics and Music 5:35–53. https://doi.org/10.1080/17459737.2011.573676

6) Rohrmeier M (2020) Towards a formalisation of musical rhythm. In: Proceedings of the 21st Int. Society for Music Information Retrieval Conf

7) Rohrmeier M (2020) The Syntax of Jazz Harmony: Diatonic Tonality, Phrase Structure, and Form. Music Theory and Analysis (MTA) 7:1–63. https://doi.org/10.11116/MTA.7.1.1

8) Rohrmeier M, Moss FC (2021) A formal model of extended tonal harmony. In: Proceedings of the 22nd International Society for Music Information Retrieval Conference. pp 569–578

9) Lieck R, Rohrmeier M (2021) Recursive Bayesian Networks: Generalising and Unifying Probabilistic Context-Free Grammars and Dynamic Bayesian Networks. In: Advances in Neural Information Processing Systems 34 (NeurIPS 2021)